POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Statistical and Graph-based approaches for Anomaly Detection in company networks

Supervisor
prof. Paolo GARZA

Candidate
Giacomo ZARBO

Company Tutor
Emanuele GALLO

Academic Year 2020-2021

Contents

Chapter 1

Introduction: cybersecurity in companies

In recent years, cyber-attacks have grown enormously. According to the FBI's *2020 Internet Crime Report*, the Internet Crime Complaint Center received 791.790 reports of cyber-crime in 2020: a 69% increase over 2019 alone[1]. In Italy, the *2021 Clusit Report* showed how from 2019 to 2020 cyber-attacks grew by 12%[2]. Although the *General data protection regulation*, that is the European regulation on privacy and data, is already in place, many companies in Europe are still in a precarious state in the cyber-security area. In addition, there are often users who do not understand or respect basic data security principles, such as choosing complex passwords, being wary of attachments in emails, or backing up data.

Targeted cyber-attacks on businesses can cause a significant reduction in profits and a great deal of expense to limit the damage caused by them, in addition to all the problems related to security and protection of information.
For business contexts specializing in Big Data, this issue is even more delicate, due to the fact that cyber-criminals have the ability to access massive and significant amounts of personal information through the use of advanced technologies that are becoming very difficult to fight. In these areas, the risk of a data breach or insider threat, the correct user's behavior, monitoring of the corporate network or of the mails, are all aspects of paramount importance.

On the other hand, the real challenge in recent years, has been to transform the world of Big Data from a "threat" into an "opportunity". There have been many research works trying to use the typical tools of Big Data as a defense against cyber-attacks. In particular, there have been, in fact, several case studies of solutions on *anomaly detection* for which the main purpose is to identify data, events and/or observations that deviate from normal behavior with respect to a considered dataset.

The purpose of this thesis project is to address and study anomaly detection solutions in business contexts, analyzing and studying the habitual behaviors of employees who are part of them and trying to find all those anomalies that can lead to serious risks within the company itself, whether because of any type of virus that has infected the user or because of bad intentions of an employee.

For the work two different approaches have been analyzed and experimented with which it has been tried to identify dangerous situations within a corporate network: one of *statistical* type, based on mathematical approaches and models, that applies anomaly detection algorithms over the behaviours of the employees, and one based on *graphs*, with which it has been used an ad hoc built algorithm that allows to identify new potentially malicious hosts.

Two datasets have been considered on which to carry out the experiments: a first artificial dataset, built to simulate the behavior of some users inside a company network, and a second dataset, granted by an important insurance company, which provides data from a log proxy.

The *statistical* approach of the project tries to make a classification of all connections that business users make to hosts (labeling them as *malicious* or *not*) using mathematical techniques and models. Specifically, *99th-Percentile*, *Z-Score* and *Inter-Quartile Range* approaches were used, and anomaly detection algorithms

applied accordingly. The use of this approach is daily, and also relying on *Reinforcement Learning* mechanisms, it tries to understand which users may be dangerous inside the company.

The second solution is based on *graphs* and tries to find malicious hosts whose classification label is not known a priori. For the experiment, through *Spark* and *GraphX* frameworks, a graph is constructed in which the nodes are represented by "users" and "hosts" while the edges are represented by a "connection" from a user to a host. The edge, in addition to containing the information of the source node and the destination node, also has all the relevant informations about that specific connection (such as timestamp, payload, HTTP status, etc.). The algorithm is daily, and through a system of loading and unloading of scores between nodes, allows the classification of those nodes whose labels are not known a priori.

Chapter 2

Anomaly detection: state of the art

Today there are many algorithms identified, studied and tested in the field of *anomaly detection*, also known as outlier detection. It is of great interest in different areas, such as business or security, as it allows to identify different types of business opportunities or critical incidents in advance. An anomaly is any type of deviation, by a data, from the entire population of which it is part and of which a certain "normal" behavior is considered. Anomaly is anything that is unexpected with respect to a specific behavioral pattern. Since this thesis work experiments with two types of approaches to anomaly detection, namely a more traditional approach based on mathematical and statistical models and an innovative approach based on graphs, the state of the art in these two areas will be discussed below.

2.1 Statistical algorithms

The statistical approaches for anomaly detection are varied: there are simple statistical methods for which to determine an anomaly but also sophisticated Machine Learning algorithms that can classify a data-point as anomalous. The state of the art of statistical approaches for anomaly detection is discussed starting with the simplest mechanisms and progressing to more sophisticated methods.

2.1.1 Statistical approaches

One of the simplest statistical approach for anomaly detection is the *Z-Score*[3]. In a statistical distribution, it indicates how much a certain data-point deviates from the rest of the data.
The formula is:

$$Z = \frac{X - \mu}{\sigma} \tag{2.1}$$

where X is the considered datapoint, μ is the mean of the train dataset and σ is its standard deviation, and mathematically represents how many standard deviations the test point is far from the mean. A Z-Score equal to 0 represents the mean itself, while a Z-Score equal to 3, for example, indicates that the point is three times the standard deviation away from the mean. Having chosen a certain threshold (a good rule of thumb is 2.5 or 3), if the absolute value of the Z-Score considered is greater than it, then the data can be considered an outlier.

Another method widely used in the literature is *Winsorization*[4], whereby any value lower than the *1st-Percentile* (i.e., that value below which 1% of the population is contained) is considered an outlier and automatically raised to the value of the 1st-Percentile. The same logic applies to values above the *99th-Percentile*, which are considered anomalies and automatically lowered to the value of the 99th-Percentile itself.

Given a certain distribution, the *median* is defined as the value of the statistical units that is in the middle of the distribution and that divides the distribution into two equal parts. The further median of one of these two parts, instead, is called *quartile*. Having obtained, therefore, a region by using three quartiles (of which the second is the median) and divided into four sub-regions, the *Inter-Quartile Range*[5] is defined as the difference between the *3rd-Quartile* and the *1st-Quartile*, as defined in the formula:

$$IQR = Q3 - Q1 \tag{2.2}$$

If a datapoint is much smaller than the first quartile or much larger than the

third quartile, it can be classified as an outlier.

The *boxplot* is a graphical representation of the further enhanced IQR. Generally the boxplot has a length equal to the IQR, while its "whiskers" have a length of $1.5 \times IQR$. This means that any point lower than $Q1 - 1.5 \times IQR$ or higher than $Q3 + 1.5 \times IQR$ is considered an outlier.

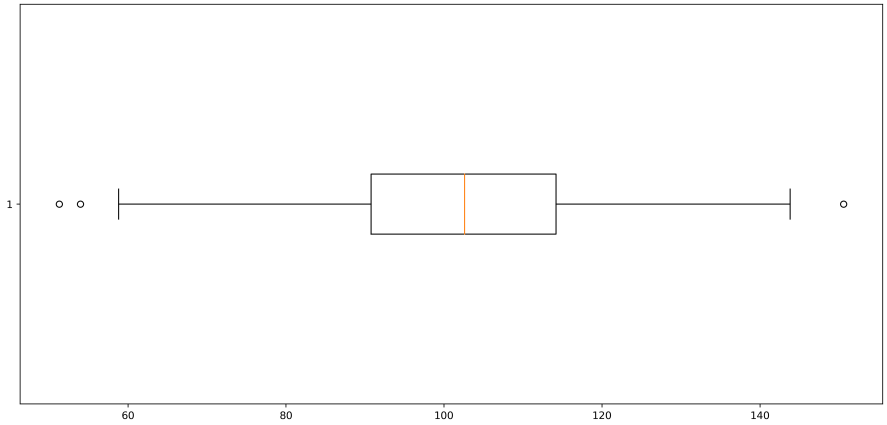


Figure 2.1: Boxplot of a random normal distribution with $\mu = 100$ and $\sigma = 20$.

2.1.2 Machine Learning algorithm approaches

Regarding Machine Learning algorithm approaches, *K-Nearest Neighbors* and *Support-Vector Machines* are certainly worth mentioning.

With the K-NN algorithm[6], a data-point is classified by a plurality vote of its neighbors, and it is assigned to the most common class among its K nearest neighbors, with K as a positive integer. If K = 1, for example, the object is simply assigned to the class of that single nearest neighbor. In this case, the anomaly detection technique, involves the use of the distances of the K-NN algorithm as an index of anomaly of a certain data-point. The outlier score of an object is given by the distance to its K-nearest neighbor. Obviously this method is very sensitive

to the parameter K: if it is too small, then only a small set of data-points can determine which are outliers, and, if it is too large, then many data-points that are part of a cluster could be outliers.

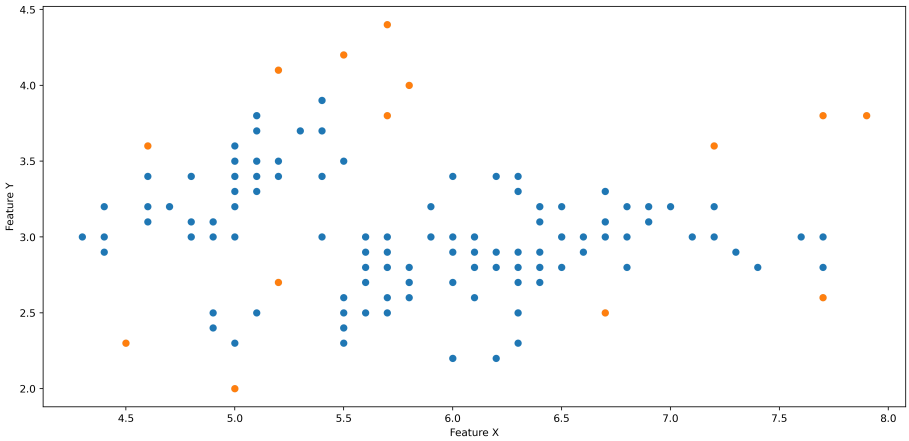


Figure 2.2: Example of anomaly detection with K-NN when K = 3. The outliers are the orange datapoints.

The other Machine Learning algorithm often used for the task of the anomaly detection is the Support-Vector Machine[7]. Generally, the SVM algorithm is used for classification problems because it uses "hyperplanes" to separate one class of data from another. However, there are many cases in which the algorithm is used on data belonging to a single class, to study their "normal" behavior and thus identify any outliers. Therefore, the algorithm takes the name of *OneClass Support-Vector Machine*[8] but with the definition of the new hyperparameter ν, which determines the percentage of data to be considered as outliers.

2.2 Graphs algorithms

In literature, there are many papers and studies done on graph algorithms in the field of anomaly detection and they can surely be divided into categories, but for the majority of these works, the definition of anomaly remains the same: an object within a dataset is defined as anomalous if its score (which can be the probability

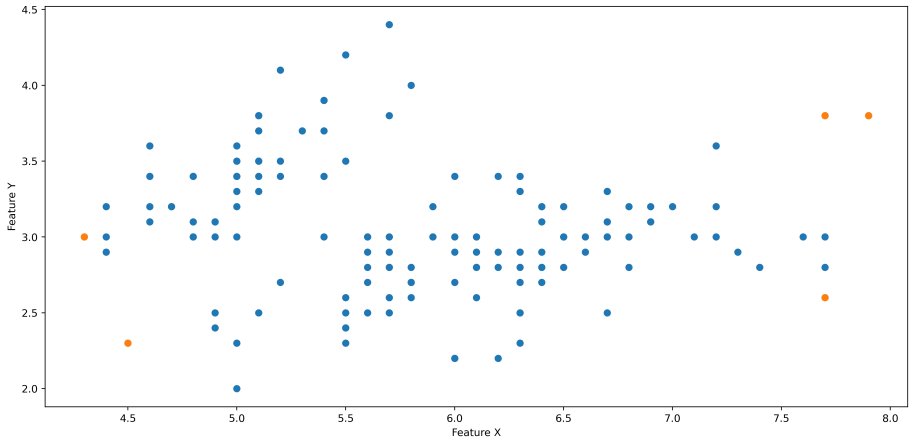


Figure 2.3: Example of anomaly detection with OneClassSVM when $\nu = 0.03$. The outliers are the orange datapoints.

of being present, or which represents its rarity) exceeds a certain threshold defined in advance.

The classification of graph algorithms for anomaly detection can be done as follows:

- Anomaly detection with static graphs;
- Anomaly detection with dynamic graphs.

2.2.1 Anomaly detection with static graphs

A *static graph* is a fixed sequence of nodes and edges, that do not evolve over time. In the current state of the art, with static approaches, a distinction must be made between two different types of graphs: the *plain graphs* or the *attributed graphs*. A plain graph is a graph that contains only nodes and edges between nodes, nothing else. An attributed graph, on the other hand, is a plain graph that contains attributes associated with nodes or edges. Therefore, the problem of anomaly detection for static graphs can be summarized in trying to find nodes or edges (or substructures of them, i.e. subgraphs) that deviate in some way from

the various patterns observed within the graph.

Anomaly detection with static plain graphs

For the graphs defined plain, the only information that can be had is the architecture of the graph itself: for this the two best procedures for the identification of anomalies are the identification of *community-based patterns* or *structure-based patterns*.

For community based patterns, the work done was to find nodes that are strictly "close" to each other, i.e. very dense groups of connected nodes that thus create a "community", and identify those nodes that connect to other communities, called *bridge-nodes*. Sun et al.[9], in their experiment, divided the task into two sub-problems: finding a community given a certain node and estimating how much a certain node can be a bridge-node. For the first sub-problem, the authors used a custom Page Rank algorithm, while for the second, the results obtained were aggregated and averaged to create a model that measures a certain normality score for nodes. Nodes with very low normality scores, therefore, have neighbors with low pairwise closeness to each other.

Other work to identify anomalies based on graph communities was conducted by Tong and Lin[10], through a mathematical procedure of matrix factorization. The factorization of a matrix X is defined as $X = Y \cdot Z + R$, where Y and Z are the low rank factors and R is the residual matrix. In the non-negative matrix factorization, there are then non-negativity constraints on Y and Z that allow communities to be found. Here they try to force the non-negativity constraint on the residual matrix R so that anomalous connections can be found.

For the structure-based models, the experiments done sought both to extract information about the centralities of the graphs and to quantify the closeness of certain nodes so as to identify certain links. In the first case, works proposed by Akoglu et al.[11] or Henderson et al.[12] extracted information from egonets (a subgraph that includes a node, the "ego", its first neighbors, the "alters", and

connections to them, as shown in Figure 2.4) and found patterns based on these. The informations extracted from the various egonets considered, such as the sum of the weights of the edges or the number of triangles formed are easy to compute and interpret. The egonets, then, studied in pairs, have allowed the identification of various patterns considered as "normal". From these, mathematical models based on power law have been constructed and, for each egonet, a deviation from a pattern has been calculated as the distance from that particular power-law distribution. Each egonet with a particular deviation received an anomaly score.

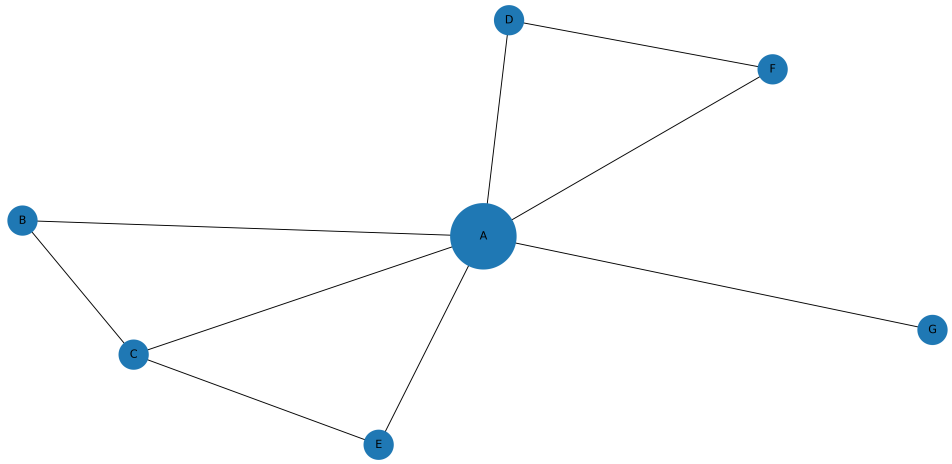


Figure 2.4: Example of an egonet for node A.

In the second case, the work of Brin and Page[13], led to a cornerstone of algorithms applicable to graphs: *PageRank*. PageRank is based on the jump from one node to another, which can occur with the same probability for each neighbor node ($1/d$ where d is the rank of the node under consideration). The probability distribution that is created, then, is used to make a ranking of nodes, based on their relevance.
An evolution of PageRank, studied by Haveliwala[14], is the *Personalized PageRank*, for which taken into account a node n and a probability parameter α it is possible to proceed with the path starting from node n, so that at each step, when it is on node m, it chooses one of its neighbors with equal probability $(1-\alpha) = dm$ and returns to the starting node n with probability α. The PPR score of a node x with respect to the starting node n is understood as the stationary probability and

represents the measure of the proximity of the node x with respect to the starting node n.

Anomaly detection with static attributed graphs

Attributed graphs are graphs with more information than plain graphs. Algorithms aimed at this type of graphs, therefore, take into account not only their structure but also the attributes that provide additional informations. As in the case of plain graphs, even here it is possible have a subdivision of algorithms for the identification of communities or structures.

In the first case, for the identification of communities, it is possible to take into account the attributes that the graph now contains. Gao et al.[15], in their work, first divided this operation into three parts: the *global outlier detection* that considers only the attributes of nodes, the *local outlier detection* that considers the attributes of neighboring nodes and the *structural outlier detection* that considers only the edges and they have developed a model to find communities within graphs but also anomalous communities between them. An evolution of this technique has been done by Muller et al.[16], with the intuition that too complex anomalies can be detected only by studying a subset of the attributes (or a subgraph). Their algorithm also has the peculiarity of quantifying, through a precise parameter, the degree of anomaly of a node, in addition to making the pure classification into anomalous or not.

In the second case, the approach takes into account not only the structure of the graph itself, but also its attributes. Noble and Cook[17] were among the first to split the problem into two parts: finding anomalous subgraphs within the graph and finding anomalous subgraphs between sets of subgraphs, where the attributes of nodes and edges are also taken into account. For the first problem, their main insight was to find an "inverse" measure of what defines the "best substructures" (i.e., the most frequent substructures): the *Minimum Description Length*. For the second problem, in a similar fashion, they tried to find a measure that would tend to penalize subgraphs with few substructures in common, increasing their degree

of anomaly.

2.2.2 Anomaly detection with dynamic graphs

The field of the anomaly detection in the world of dynamic graphs, refers to events that can change in the time, that is, given a certain temporal cadence, to a succession in the course of the time of a whole of static graphs, one for every temporal slot. So the main purpose of this branch, is to study the evolution over time of a graph, analyzing the changes in structure and the occurrence of any anomalies, referring to the normality of the past.

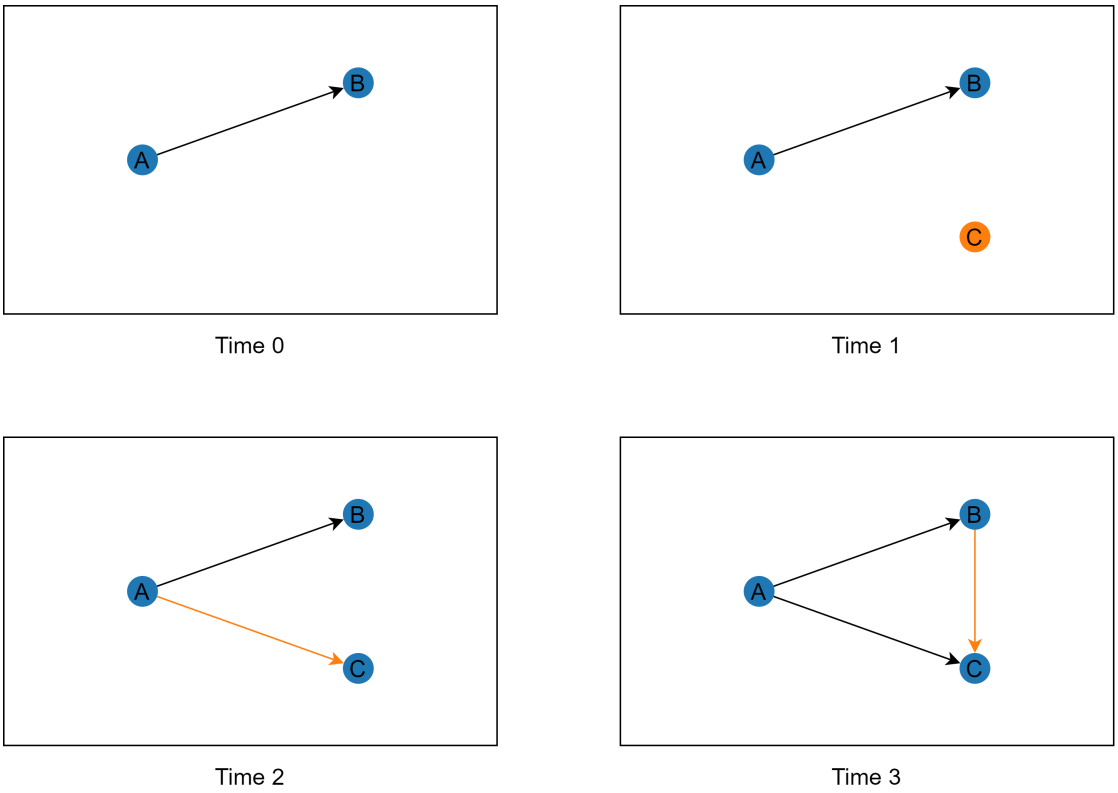


Figure 2.5: Example of a dynamic graph.

A macro distinction that can be made on algorithms that operate on dynamic graphs is as follows:

- Feature-based algorithms;

- Decomposition-based algorithms;
- Community or clustering-based algorithms;
- Window-based algorithms.

Feature-based algorithms

Similar graphs have similar properties, and that's the idea behind these kinds of approaches. The basic mechanism consists, for each graph belonging to a certain time interval, in extracting the most relevant information, comparing it with those of previous time intervals with some measure of similarity and classifying it as anomalous or not when a certain threshold is exceeded. Some of these measures of similarity have been studied by Bunke[18] and Shoubridge[19]: for example, the measure of *Maximum Common Subgraph (MCS)* which is the largest isomorphic graph to two subgraphs, or the *Graph Edit Distance (GED)* which indicates the number of topological operations required to transform a graph in another, or even the *Hamming distance* that counts the number of entries of the matrices of adjacency of the graph.

Decomposition-based algorithms

The decomposition-based event approach detects temporal anomalies by making use of tensor and matrix decompositions, appropriately analyzing their eigenvalues or eigenvectors. For example, the method of Idé and Kashima[20], extracts the eigenvectors of the adjacency matrix for each graph and then, applying the *Singular Value Decomposition* based on past graphs, finds the "typical activity" and calculates the difference between the current vector and the vector representing typical activity. Or Ishibashi's work[21], which proposes the creation of an adjacency matrix with cells that contain the similarity information between the connections in the graph based on the destination hosts they have in common.

Community or clustering-based algorithms

For these types of approaches, the main goal is to monitor the communities and clusters within the graph over time in order to detect anomalous behavior. Sun et

al[22] defined *GraphScope*, an MDL-based algorithm that seeks to discover subsets of nodes and detect changes within them. A subset of nodes consists of having a set of "similar" nodes, i.e. with an adjacency matrix that when divided increases its coding cost. The algorithm tries to derive the best source and destination subsets until further division leads to a decrease in encoding cost.
Peel and Clauset[23], on the other hand, worked on GHRG, or *Generalized Hierarchical Random Graphs*. This is a model for decomposing graph nodes into a set of nested groups, whose relationships can be described with a dendogram. This type of representation allows to represent the communities within the graph on multiple levels of scale, starting from larger clusters down to smaller clusters. Significant changes in the pattern imply the appearance of an anomaly.

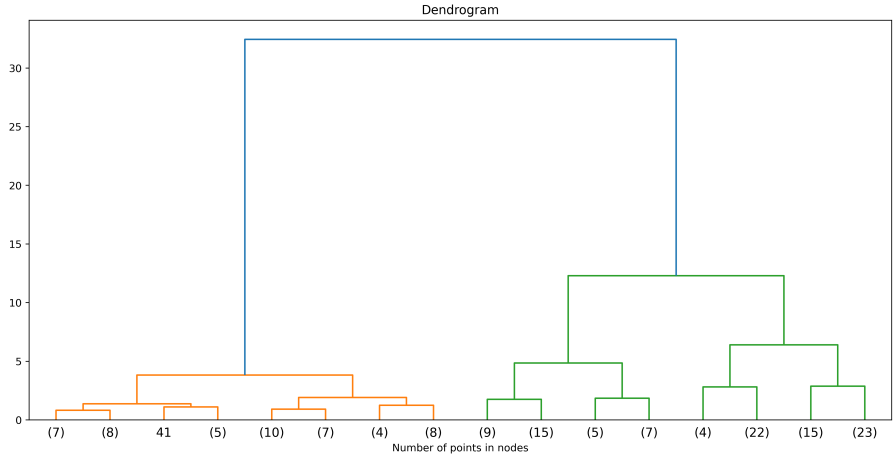


Figure 2.6: Example of a dendogram.

Window-based algorithms

For this category, the approach is to try to detect anomalies based on a certain time window of sequence of graphs, defining it as "normal" behavior. The graph taken into consideration is, then, compared with the one built on the time window, so as to identify any anomalies. First, Priebe et al[24] applied moving-window statistics to identify anomalous behavior of the examined graph with respect to its past. The procedure consists of evaluating statistics local to the time window,

and considering the maximum statistic detected as the *scan statistic*. Any scan statistic that exceeds a certain threshold labels the corresponding time frame as an outlier. The local statistics can be the number of edges, for each node, toward neighboring k-nodes or the number of paths and stars (the set of edges whose source is a given central node). In more recent years, Mongiovi et al.[25] have transformed the problem of anomaly detection in dynamic graphs into an NP-HARD problem, through the search of the *Heaviest Dynamic Subgraph (HDS)*, i.e. the search, for all the weighted graphs of the time window, of the lowest p-value, index of anomaly, considering the empirical distribution of the weights of the edges. This mechanism brings two detections: one in the space dimension and one in the time dimension: the detection of the subgraph with the highest anomaly score in the whole graph, and the detection of the temporal interval that has the highest anomaly score for a considered subgraph.

Chapter 3

First approach: a statistical method to detect anomalies in business networks

In recent years, and more and more frequently, the topic of cybersecurity, which affects any type of entity or company but also the individual, has been brought to everyone's attention. Its importance derives from the fact that nowadays cyber-criminals commit crimes increasingly insistently and with even more sophisticated methods. This area affects everyone, including all companies that operate in any industry and need to safeguard their profits and expenses.

This thesis work, therefore, aims to study and experiment with cybersecurity techniques, in business contexts, but with data-intensive approaches. In fact, it is possible to try to look at the topic of Big Data not as a possible threat, but as an opportunity (or tool) to build effective security methods.

The first approach used in this experiment is purely statistical, as it makes use of mathematical models, built on the basis of a large amount of data made available, and with which it is possible to identify possible threats in the context of a corporate network.

In particular, the aim is to discover possible users infected by malicious viruses, studying their habitual behaviour and evaluating possible anomalies.

3.1 Input data

Due to the lack of datasets suitable for a work of this type, the decision was to test different approaches on a dataset built specifically for this purpose and, subsequently, test evolutions of these approaches also on a real one.

3.1.1 The artificial dataset

The dataset artificially constructed has the purpose to simulate the connections of users inside a company network towards the hosts of the entire internet network, and this is the behavior of a *Proxy Log Server*.

The dataset is built on files with `.csv` extension divided according to the day considered. In particular *50 days* are considered (starting from *day 0* up to *day 49*), in order to test the algorithm to its maximum potential and evaluate the results on more days. Therefore, a `.csv` file is constructed for every day.

For each file, each line represents a connection, and for each connection the following fields are included:

- the `time` at which the connection occurs, expressed simply by an integer because the dataset is already divided for each day;
- the `user`, a unique ID of the user who initiates the connection;
- the `host`, a unique ID for the destination host;
- the `URL` of the connection, held up to the domain name;
- the `status`, i.e. the HTTP status code of the connection;
- the `payload` of the connection (in Bytes);

- the `label` that identifies if the connection is malicious or not.

In the construction algorithm, a *User Class* is defined, which represents the entity of an employee within a company. This class is characterized by having, with a certain *degree of randomness*, a habitual behavior, one for each user, so one for each instance of the class.

To get a fairly reliable simulation, in fact, the behavior of whichever user inside the company, is described with this class, like a real person, with the own habits, the own interests (therefore more inclined to visit certain website rather than others), an own randomness in to carry out not habitual actions. So, for 50 days, the construction algorithm creates a dataset for each day, in which are described all the connections that all users initiate during that day, but each connection is characterized by peculiarities of the user who initiated it.

Obviously the number of connections per user is not always the same, just as there are not always only the same hosts to which a user connects: as written, each user has the possibility, with a certain degree of randomness, to make a connection to a "habitual" site, to make a connection that is not "habitual" (to a random host), to make no connection at all or to make a habitual connection but with different parameters (such as HTTP Status or payload).

This logic serves to improve the algorithm itself: it is always good to try to include a certain degree of unpredictability, otherwise the algorithm would be completely useless because it would train on data that would never represent a real case (or it would be a case too easy to study).

For the construction of the dataset, after *30 days*, a new feature of the algorithm comes into play: the `try_hacking` function.
This function, called for each user at the time they need to make a connection, tries to hack the user. Obviously the probability of this happening is very low, but it is also proportional to the amount of users that the network contains, so as to simulate an authentic and truthful network.
If a user has the misfortune of being hacked, then he will start to have a series

of behaviors that deviate from his normality, to simulate a *virus* acting in the background.

The virus causes the user to make connections to new malicious hosts with attributes related to various aspects: for example, it might have a very large payload because it tries to do exfiltration of a large amount of data, or it might have a payload that always remains constant over time because the virus splits the data before doing the exfiltration. Or even, try to make random connections to non-existing servers, thus getting HTTP statuses always equal to 404.

On the other hand, in a real case, thanks to the work done by the security department of a company, it is not said that a user remains hacked for a long time. In the dataset, therefore, every hacked user has a certain probability to be restored starting from the next day. In fact, the evaluation of users, whether they are hacked or not, is done daily.

Another feature of the algorithm is the possibility, with a certain degree of probability, to *propagate the virus* to the rest of the users of the company if one of them is infected: this represents a real-world case, as users in a company are always communicating with each other. The consequence is that the more infected users there are, the easier it is for the virus to spread within the network.

3.1.2 The real dataset

The other dataset used represents data from a *Proxy Log Server* of a major insurance company.

As for the artificial dataset, the real one consists of 25 subdatasets, each of which represents a day of the year in which the users of the company operates, and in this case *25 days* are chosen, each following the other, from the second quarter of the year 2021, therefore very recent data.

However, it is necessary to specify that although the data are real, the latest

health conditions in our country, due to the Covid-19 pandemic, have led many companies to make their employees work from home, therefore often not connected to the company network. This has affected quite a lot the connections that users have made within the company, in terms of quality and quantity.

As mentioned, the dataset is composed of 25 `.csv` file, one for each day, all representing connections from users to hosts outside the company network.

Every file is composed by the fields:

- `devicetime`, the timestamp at which the connection is started;
- `cip`, the identifier of the client that performed the connection to the host;
- `dst`, the identifier of the host to which the connection is made;
- `cshost`, the name of the host;
- `csuri`, the url of the site to which the connection is made;
- `scstatus`, the HTTP status code of the connection;
- `csbytes`, number of bytes sent by the user to the host;
- `filtercategory`, the "category" given by the Proxy Log Server to the host of the connection;
- `xvirusid`, the virus id, if the host is already known to be malicious, `"-"` otherwise;

For security and privacy reasons, the `cip`, `sip`, `dst`, `cshost` and `csuri` fields were first anonymized and then subsequently mapped to unique numeric values so that they could be used for subsequent proceedings.

The maliciousness of a connection, and thus of the corresponding host, is made known by the `filtercategory` and `xvirusid` fields. Therefore, through appropriate analysis, a new `label` field is created, which has the values `MAL` if the connection is among filter categories that can be considered as malicious or if the virus id is known, or `OK` otherwise.

3.2 Feature engineering

With the dataset available, the first task performed is the *Feature Engineering*: the purpose is to extract useful information for the algorithm, starting from raw data, taken as they are made available.

This procedure is done mainly for two reasons. First, to consider only relevant data, discarding those less useful for the purposes of the experiment. Secondly, to know the habits of individual users within the company, therefore it is necessary to extract from the proxy logs all the information that can establish certain behavioral trends. This is done by taking into consideration the "raw" information of the proxy log and aggregating it, it will be explained later how, in such a way as to create "quantitative" features, since these better represent the habits of users and better highlight any anomalies. For example, it is much easier to think about the habits of a user knowing not which sites he usually visits (a too onerous calculation if the number of users in the company is very high), but how many sites he visits the most, and at the same time it is much easier to detect an anomaly in the number of sites visited during the day, rather than considering any site he has never visited before.

Thus, the feature engineering operation is done *for each user* and *for each day*: in this way the information collected will be considered as belonging to that user, in order to be able to understand the individual behavior, day by day.

Then, the following new features are created:

- `cnt_domain` representing the number of distinct top-level domains of the URL belonging to connections;
- `cnt_status_2xx` representing the number of connections with $200 <= payload < 300$;
- `cnt_status_4xx` representing the number of connections with $400 <= payload < 500$;

- `cnt_status_5xx` representing the number of connections with $500 <= payload < 600$;
- `cnt_payload` which indicates the number of distinct payloads belonging to connections;
- `cnt_host` which represents the number of distinct hosts the user connected to during the day, extracted as the number of distinct hostIDs;
- `cnt_URL` representing the number of distinct URLs the user connected to during the day;
- `avg_payload` which indicates the average (in Bytes) of payloads belonging to connections made on that specific day by the user;
- `unique_payload` that indicates the number of connections that have had as payload the most frequent payload used by the user for that day; basically it's calculated how many times each payload is used in that day and the count of the payload with the highest count is taken into account;
- `label` which represents the label value the user had for that day under consideration: *HACK* if it has at least one malicious connection, *OK* otherwise.

All the values, collected for each user, are arranged within a Pandas dataframe (with row index the user considered and the daily attributes arranged in columns), so as to facilitate the subsequent calculation of statistics.

As it can be seen from the Figures 3.1 and 3.2, the histograms, on a daily basis, of a user of the considered dataset are reported. On the x-axis there are the *days*, while on the y-axis there are the *frequencies* (or the *means*) and they represent the values of the features just created. Each graph also has horizontal lines that represent the *average* (the orange line) for that feature for all considered days, the *median* (the red line), the *25th Percentile* and the *75th Percentile* (the green lines). Knowing that in some days the user has been hacked (or it has had a different behaviour with respect to his normal one), it is possible to immediately notice how some trends change abruptly, represented by very high or very low peaks of the histogram bars.

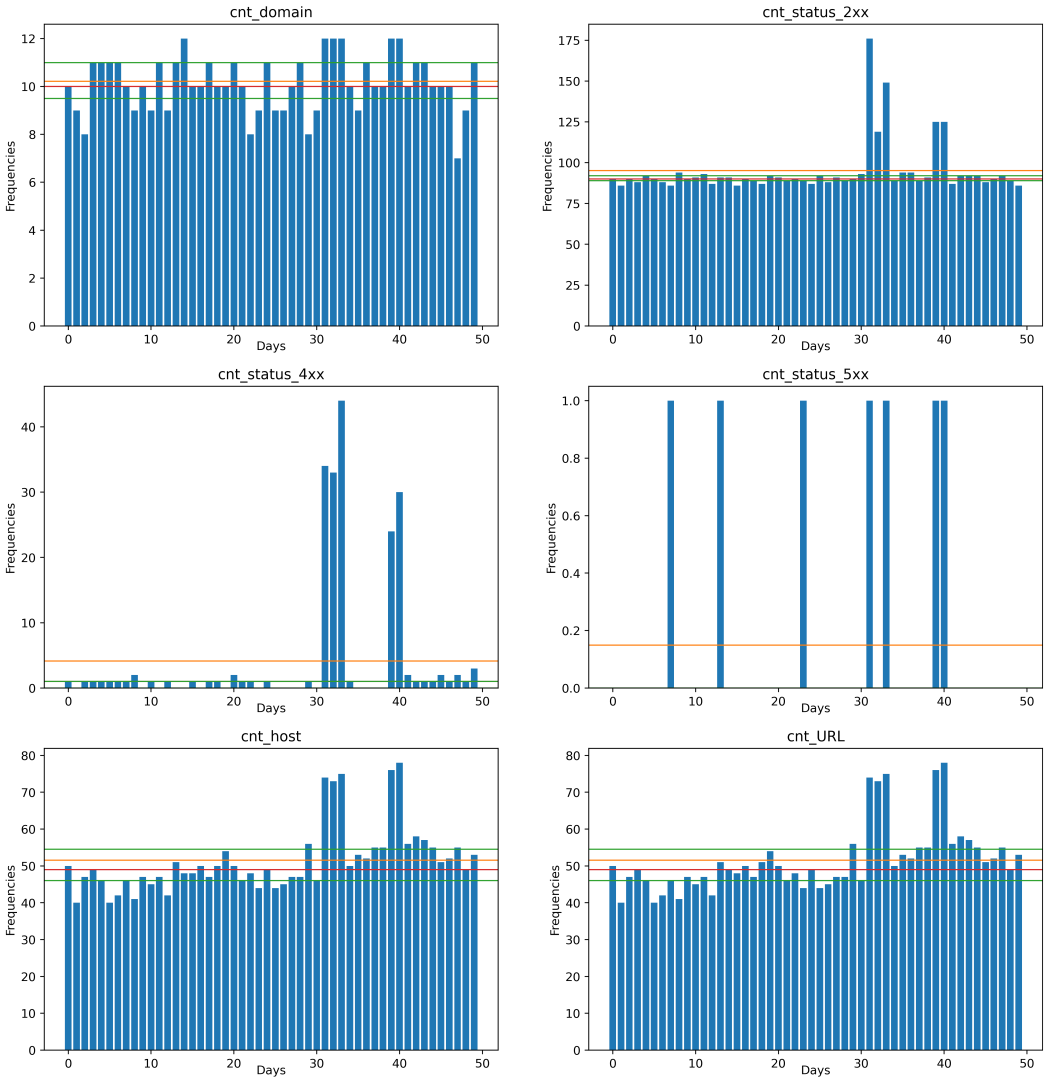


Figure 3.1: Histograms of features created for a user taken as an example from the artificial dataset.

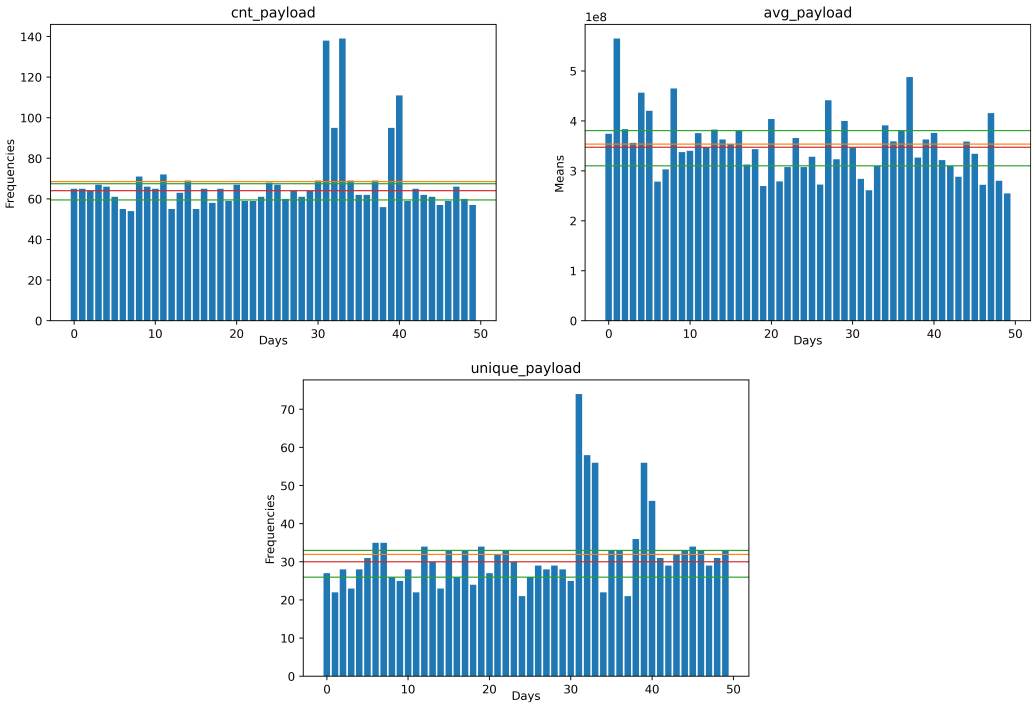


Figure 3.2: Histograms of features created for a user taken as an example from the artificial dataset. These features are all extracted from the starting raw feature *payload*.

3.3 Anomaly Detection: recognition of hacked users

For the detection of hacked users within the corporate network, three different anomaly detection techniques are used on the artificial dataset, which are then evaluated, and the best of these is chosen for continuation of the experiment on the artificial and real ones.

The three approaches used are:

- Percentile-99;
- Z-Score;

Algorithm 1 Moving window algorithm.

```
1: X ← 0;
2: while X ≠ 20 do
3:     Train from X to X + 29;
4:     Test on X + 30;
5:     X ← X + 1;
6: end while
```

- Inter-Quartile Range.

3.3.1 Percentile-99

The first technique used is the *Percentile-99*: any test value greater than the Percentile-99 of its train distribution, is assumed to be an outlier.

For each user, and for each feature, the values of *30 days* are considered as *train set* (at the first iteration they are the values from *day 0* to *day 29*), of which Percentile-99 is calculated, while the values of the day immediately following are considered as *test set* (at the first iteration they are the values of *day 30*), so they are compared with the Percentile-99 and classified as anomalous or not.
The algorithm is iterative, in fact it tries to use *30 days* available for the training phase and evaluate the day immediately following, advancing one day at a time, like a moving window. Tests, then, are done on *20 days*.
The pseudo-code is shown at Algorithm 1.

Since a single feature anomaly is not necessarily an indication that a user has been infected by a virus, the experiment continues to identify the *total number of anomalies* to be taken into account in order to classify a user as hacked. So, for each user, for each day, the number of anomalies of all features is collected (a single feature has 1 if it finds the anomaly, 0 otherwise).

Then, the algorithm proceeds to *classify* a user as hacked or not choosing a threshold for the number of anomalies: a user can be considered hacked if, for a given day, there are a total of 3, 4, 5 or 6 anomalies in his behavior. Different

results obtained will be then discussed and evaluated in Chapter 6.

3.3.2 Z-Score

The second technique used is the *Z-Score*. As in the previous case, the algorithm has a moving window mechanism: it is trained considering *30 days* and it is tested on the following day (pseudo-code is Algorithm 1), for *20 days*. In this case, however, for each user, for each feature and for the *30 days* considered by the moving window, the train data are used to compute the *mean* and the *standard deviation* (admissible values since the features contain only numerical variables).

Subsequently, in the testing phase, the test values taken into consideration are *normalized*: for each user and for each feature, the difference is made with the mean of the train values and the ratio with the standard deviation, both found previously, as shown in Equation 2.1. The decision for which a data of a feature can be considered as anomaly or not is made with a rule of thumb widely used today: a value is considered outlier if its Z-Score is greater than 2.5 (i.e. if it is 2.5 standard deviation far from the mean). Then, the total number of anomalies are collected.

For the classification, an user is considered as hacked if the number of anomalies occurring in the same day (the test day) is greater than or equal to a certain threshold. In this case the experiment is done with four different thresholds: 3, 4, 5 and 6. Different results will be discussed and evaluated in Chapter 6.

3.3.3 Inter-Quartile Range

The last technique used is the *Inter-Quartile Range*. As in the previous cases, the algorithm proceeds with a moving window, training on *30 days* taken as a reference and testing on the following day, testing *20 days* in total (pseudo-code is Algorithm 1).

Therefore, for each user and for each feature, the *Percentile-25* and *Percentile-75* values, i.e., the first (Q1) and third (Q3) quartile values, are calculated on train

set. Next, the two limiting values considered for outliers detection are:

- $lower\_limit = Q1 - 1.5 \times IQR$
- $upper\_limit = Q3 + 1.5 \times IQR$

that correspond to the tips of the "whiskers" of a boxplot diagram. During the testing phase, therefore, any value lower than *lower_limit* or higher than *upper_limit* will be considered as an outlier.

As done for the previous two techniques, a user is considered hacked on a certain day if the number of his anomalies on that day is 3, 4, 5 or 6. Different results will be reported in Chapter 6.

3.3.4 Results

The results obtained from experiments performed on the artificial dataset, which will then be discussed in the Chapter 6, lead to consider the Inter-Quartile Range technique as the one that provides the best classification, so this will be used for the experiments explained later, that make use of both the artificial and the real dataset.

It must be mentioned, however, how these approaches are extremely flexible and scalable, in fact the ability to adapt to the behavior of a single user to identify any anomalous deviations is strong and, through feature engineering, it is very easy to take into account other features (e.g. add new ones) if the dataset studied is different.

3.4 Reinforcement Learning

Reinforcement Learning is a Machine Learning technique that aims to create autonomous agents capable of choosing actions to accomplish certain goals through interaction with the environment in which they are immersed.

Reinforcement Learning approaches can be made simpler, hence more static, if it is assumed that a data analysis team and a security team are ready to help the algorithm itself, but for this thesis work, the choice was to move towards dynamic approaches, with the algorithm "changing" depending on the results obtained.

Indeed, the experiment proceeds using three different Reinforcement Learning approaches, all sharing the fact that, day by day, they evaluate whether to trigger the "reinforcement" or not, trying to adjust a series of parameters by referring to the results obtained in the previous days. The details of each approach are discussed below.

3.4.1 First approach

Considering the use of IQR, that previously has given the best results, it is possible to assert that the parameters that determine whether or not an anomaly is spotted are Q1, Q3, and the constant 1.5, since they are present within the equations:

- $lower\_limit = Q1 - 1.5 \times IQR$;
- $upper\_limit = Q3 + 1.5 \times IQR$;

where $IQR = Q3 - Q1$.

Since Q1 and Q3 are the parameters dependent on the values of the features taken into consideration, and therefore not modifiable, it is possible to think that the parameter that can undergo a certain change is the remaining constant 1.5[26]. Now this parameter is no longer constant and is called C.
The two equations become:

- $lower\_limit = Q1 - C \times IQR$;
- $upper\_limit = Q3 + C \times IQR$.

With the same previous logic, then any value above *upper_limit* or below *lower_limit* is considered as anomaly. The new Reinforcement Learning algorithm will then attempt to change the C parameter based on the previous days'

results.

In addition to this type of Reinforcement Learning approach, there is another higher-level one that decreases or increases the *number of anomalies*, from now on called *threshold*, that must be taken into account in order to classify the user as hacked or not. This comes into play when the previous method leads to having C values that are too low or too high.

So, finally, there are *two levels* of reinforcement: one to classify the value of a feature as an anomaly or not, and one, of higher level, to decide the threshold of the number of anomalies to be considered in order to classify a user as hacked or not.

Algorithm 2 Moving window algorithm with Reinforcement Learning for the artificial dataset.

```
1: X ← 0;
2: while X ≠ 20 do
3:     Train from X to X + 29;
4:     Test on X + 30;
5:     Evaluate Reinforcement Tilt;
6:     X ← X + 1;
7: end while
```

Algorithm 3 Moving window algorithm with Reinforcement Learning for the real dataset.

```
1: X ← 0;
2: while X ≠ 10 do
3:     Train from X to X + 14;
4:     Test on X + 15;
5:     Evaluate Reinforcement Tilt;
6:     X ← X + 1;
7: end while
```

This first algorithm starts with the threshold equal to 5 and the C parameter equal to 2.0. As shown in pseudo-code of Algorithm 2 and Algorithm 3, the training phase is done with a moving-window over *30 days* for the artificial dataset

and over *15 days* for the real one, advancing day by day, in order to evaluate the behaviour of the user, and then the testing phase is executed, to the day following the last day of the moving window considered, respectively for *20* and *10 days* in total. The novelty, with respect the approaches previously explained, lies in the fact that reinforcement, day by day, can be tilted or not. To tilt the reinforcement, a comparison is made between the *accuracy* calculated on the day under consideration and the accuracy calculated on the previous day (the accuracy is the ratio of the number of correct predictions to the total number of observations but it will be explained better in Chapter 6).

If the accuracy of the test day is greater than or equal to the accuracy of the previous one, the reinforcement is not activated, otherwise it is, because it means that there has been a decrease in the goodness of the model.
The reinforcement happens in this way: given the number of *False Positives* (FP, number of users incorrectly classified as hacked) and *False Negatives* (FN, number of users incorrectly classified as non-hacked) found on that day, if $FP \geq FN$, then the C parameter is increased by 0.1, otherwise, if $FP < FN$, then the C parameter is decreased by 0.1.

As also explained in Figure 3.3, the logic behind this choice lies in the fact that if the number of False Positives is greater than the number of False Negatives, it means that the algorithm has been not very "restrictive" in classifying the anomalies, so it is possible to increase the C parameter, or widen the whiskers of the boxplot diagram. If instead the number of the false positives is smaller of that of the false negatives, it means that the algorithm has been, to the contrary, too much "restrictive" and it tries therefore to decrease the C parameter and to tighten the whiskers of the boxplot diagram.

It is good to understand, however, that too high or too low values of C could lead to incorrect classification by the model, if not a meaningless mechanism. In fact, it is necessary to choose some limit values for the value C: they are 1 (for which C cannot be lower) and 3 (for which C cannot be higher).

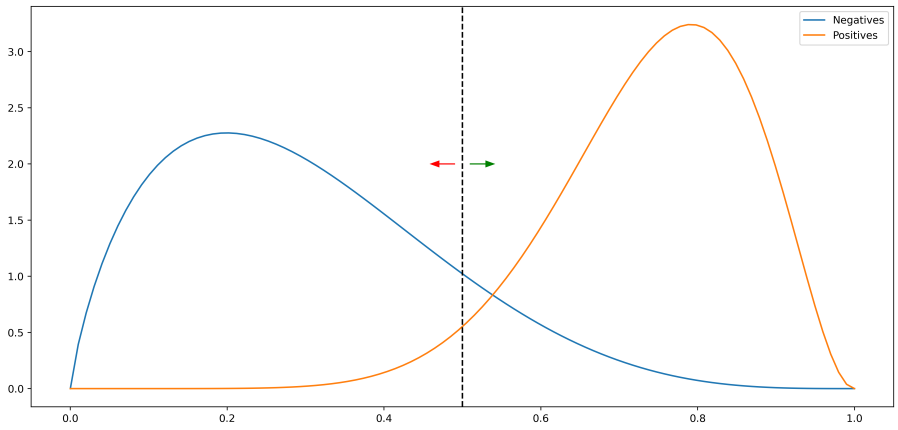


Figure 3.3: Representation of an example of a moving threshold with two probability density functions for Positives and Negatives. Elements to the right of the threshold are classified as Positives, elements to the left are classified as Negatives. Therefore, FNs are to the left, under the orange curve, and FPs are to the right, under the blue curve. If $FP \geq FN$, threshold moves to the right, otherwise moves to the left.

The *second level of reinforcement* has, instead, the following behaviour.
If C has already reached the value of 1, and the algorithm tries to decrease the value further (so the number of false negatives is greater than or equal than the number of false positives), the value of C is returned to 2 while the *threshold* is decreased by 1 (and for this the minimum value is 3). Vice versa, if C is equal to 3 and the algorithm tries to further increase the value (so the number of false positives is greater than the number of false negatives), the value of C is returned to 2 while *threshold* is increased by 1 (and for this the maximum attainable value is 6).

The logic is always the same: if the number of False Positives, day by day, is persistently greater than the number of False Negatives, i.e. the algorithm is not very "restrictive", it means that it should be considered the second level of reinforcement, and for this reason the choice is to increase the number of anomalies to be taken into account for the classification. On the contrary, if the number of False Negatives, day by day, is persistently greater than the number of False Positives,

i.e. the algorithm is too "restrictive", the second level of reinforcement should be considered, but now the choice is to decrease the number of anomalies to be taken into account for the classification.

The approach discussed and its trend for the artificial and real datasets is shown in Table 3.1 and Table 3.2 while results will be shown in Chapter 6.

Days	**Reinforcement tilt**	**C**	**Threshold**
day_1	NO	2.00	5
day_2	YES	2.00	5
day_3	NO	1.90	5
day_4	NO	1.90	5
day_5	NO	1.90	5
day_6	YES	1.90	5
day_7	NO	1.80	5
day_8	YES	1.80	5
day_9	YES	1.70	5
day_10	NO	1.60	5
day_11	YES	1.60	5
day_12	NO	1.50	5
day_13	YES	1.50	5
day_14	YES	1.40	5
day_15	NO	1.30	5
day_16	NO	1.30	5
day_17	YES	1.30	5
day_18	YES	1.20	5
day_19	NO	1.10	5
day_20	YES	1.10	5

Table 3.1: Trend of the first approach for the artificial dataset.

Days	Reinforcement tilt	C	Threshold
day_1	NO	2.00	5
day_2	YES	2.00	5
day_3	NO	2.10	5
day_4	NO	2.10	5
day_5	NO	2.10	5
day_6	NO	2.10	5
day_7	NO	2.10	5
day_8	NO	2.10	5
day_9	YES	2.10	5
day_10	YES	2.20	5

Table 3.2: Trend of the first approach for the real dataset.

3.4.2 Second approach

A second approach starts from the first, but with the assumption that it is much easier to consider as "abnormal" a behavior that results in very high feature values, rather than one that results in very low ones. This assumption finds explanation in the fact that a user, inside a business network, could, due to some unforeseen event, have less connections in comparison to his habits, but this is certainly not symptom of infection from a virus. A virus, just for its nature, tends to increase the number of connections of a user, not to decrease it.

A clarifying example is an employee who goes on vacation for two or three days: certainly he will have abnormal feature values compared to his usual behavior (for two or three days he will have practically all features values equal to 0) but this doesn't mean that the employee has definitely been hacked. This consideration, therefore, leads to a slightly different approach with IQR: now, for each feature, a certain *mechanism* for considering anomalies can be set.

For each feature, there are 3 mechanisms:

- `UP`, if only very high values, those exceeding *upper_limit*, are considered as anomalous;

- `LOW`, if it is considered as anomalous only a very low value, that less than *lower_limit*;

- `BOTH`, if both types of values are considered as anomalous.

These mechanisms, however, are static and they can be decided, for each feature, in the initial phase by a security operator. The choice of having the mechanism static, i.e. not changing over time unless a manual change is made, is due to the fact that viruses can act differently. For example, for the dataset considered, the feature `avg_payload` has a mechanism equal to `BOTH`: this is because the average payload of a hacked user could either decrease (for example in the case of a virus that makes a series of multiple connections with very low payload) or increase (for example in the case of a virus that tends to make few connections but all with very large payload).

The use of moving-window and reinforcement tilt of the previous approach remain, as shown in pseudo-code of Algorithm 2 and Algorithm 3, and the algorithm starts, the first day, with the *threshold* equal to 5 and with the C parameter equal to 2.

The experiment provides the better performance for mechanisms chosen as:

```
    feature_mechanism = {
      'cnt_domain':  'up',
      'cnt_status_2xx':  'up',
      'cnt_status_4xx':  'up',
      'cnt_status_5xx':  'up',
      'cnt_host':  'up',
      'cnt_URL': 'up',
      'cnt_payload':  'up',
      'avg_payload':  'both',
      'unique_payload':  'up'
}
```

and its trend for the artificial and real datasets is shown in Table 3.3 and Table 3.4.

Days	Reinforcement tilt	C	Threshold
day_1	NO	2.00	5
day_2	YES	2.00	5
day_3	NO	1.90	5
day_4	NO	1.90	5
day_5	NO	1.90	5
day_6	YES	1.90	5
day_7	NO	1.80	5
day_8	YES	1.80	5
day_9	YES	1.70	5
day_10	NO	1.60	5
day_11	YES	1.60	5
day_12	NO	1.50	5
day_13	YES	1.50	5
day_14	YES	1.40	5
day_15	NO	1.30	5
day_16	NO	1.30	5
day_17	YES	1.30	5
day_18	NO	1.20	5
day_19	NO	1.20	5
day_20	YES	1.20	5

Table 3.3: Trend of the second approach for the artificial dataset.

The results, instead, will be shown in Chapter 6.

3.4.3 Third approach

The last approach is similar to the first. The moving-window and the reinforcement tilt remain, as shown in pseudo-code of Algorithm 2 and Algorithm 3, but the reinforcement makes use of a weight system. The C parameter is used and, again, for a certain day, it is increased if the number of FPs is greater than or equal to the number of FNs, otherwise it is decreased.

Instead, the second level of reinforcement disappears, leaving room for a system

Days	Reinforcement tilt	C	Threshold
day_1	NO	2.00	5
day_2	YES	2.00	5
day_3	NO	2.10	5
day_4	NO	2.10	5
day_5	NO	2.10	5
day_6	NO	2.10	5
day_7	NO	2.10	5
day_8	NO	2.10	5
day_9	YES	2.10	5
day_10	YES	2.20	5

Table 3.4: Trend of the second approach for the real dataset.

of *weights* and *scores*.
For each feature considered, an initial parameter is set, that is its own weight, which will then affect the relevance of the feature itself in the classification of hacked and non-hacked users. This is done because not all the considered features have the same importance in the calculation of the user's habitual behavior.
Indeed, depending on the company's network and the different actions of the viruses that infect the users, several features considered with the same importance could lead to less accurate results.

For example, there might be a virus that infects a user and forces him to make multiple connections to a certain host with relatively low payloads: in this case the features `unique_payload` and `avg_payload` are more relevant than `cnt_status_4xx` or `cnt_status_5xx`.

Thus, for each day and for each user, the detection of an anomaly, for each feature, is always evaluated through the IQR technique and with the C parameter changing dynamically. If the anomaly is present, the weight of the feature is taken into account, for the calculation of a certain *score*. This score is the sum of all the weights associated to the features considered as anomalous and it is used for the classification of the users.
If the score is greater than a certain threshold (0.9, found experimentally for these

datasets), that day the user is classified as hacked.

Not knowing, however, which weights to give to the features, since it is too difficult to know in advance the relevance of each of them, the system uses a dynamic approach. Depending on the results of the previous day it is decided whether to raise or lower the weight of each feature.

If the accuracy of the day in question is greater than or equal to that of the previous day, there is no reinforcement, so the weight of the features remains the same and the C parameter does not change. If the accuracy of the day, instead, is less than the previous day's accuracy, reinforcement occurs. In this case, as previously mentioned, both the C parameter and the weights change. In fact, for each feature, if the number of FPs is greater than or equal to the number of FNs, the feature weight is decreased by 0.1 if is between 0 and 0.5 (0.5 included) and increased by 0.1 if it is between 0.5 and 1, and C increased by 0.1. Conversely, if the number of FPs is less than the number of FNs, the feature weight is increased by 0.1 if it is between 0 and 0.5 or decreased by 0.1 if it is between 0.5 and 1 (0.5 included), and C decreased by 0.1.

This is because, if the number of FPs is too high, it means that the algorithm is too little "restrictive" and it would try to give even more importance to the most relevant features, increasing their weights, and give less to those less relevant, decreasing their weights. If instead the number of FN is too high, then it would mean that the algorithm is too "restrictive" and it would try, therefore, to "soften" the weights among all the features, thus flattening them all around a value of 0.5.

Starting, at the first day, from weights defined as:

```
weights = {
  "cnt_domain":  0.4,
  'cnt_status_2xx':  0.5,
  'cnt_status_4xx':  0.3,
  'cnt_status_5xx':  0.3,
```

```
    'cnt_host':  0.4,
    'cnt_URL': 0.5,
    'cnt_payload':  0.3,
    'avg_payload':  0.5,
    'unique_payload':  0.3
}
```

the algorithm proceeds and its trend for the artificial and real datasets is shown in Table 3.5 and Table 3.6, with daily weight for each feature.

Days	cnt_dom	cnt_2xx	cnt_4xx	cnt_5xx	cnt_host	cnt_URL	cnt_payl	avg_payl	un_payl	Reinf. tilt	C
day_1	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_2	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_3	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_4	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_5	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_6	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_7	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	NO	2.00
day_8	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	0.30	YES	2.00
day_9	0.50	0.40	0.40	0.40	0.50	0.40	0.40	0.40	0.40	NO	1.90
day_10	0.50	0.40	0.40	0.40	0.50	0.40	0.40	0.40	0.40	NO	1.90
day_11	0.50	0.40	0.40	0.40	0.50	0.40	0.40	0.40	0.40	YES	1.90
day_12	0.40	0.50	0.50	0.50	0.40	0.50	0.50	0.50	0.50	NO	1.80
day_13	0.40	0.50	0.50	0.50	0.40	0.50	0.50	0.50	0.50	YES	1.80
day_14	0.30	0.60	0.60	0.60	0.30	0.60	0.60	0.60	0.60	NO	1.90
day_15	0.30	0.60	0.60	0.60	0.30	0.60	0.60	0.60	0.60	NO	1.90
day_16	0.30	0.60	0.60	0.60	0.30	0.60	0.60	0.60	0.60	NO	1.90
day_17	0.30	0.60	0.60	0.60	0.30	0.60	0.60	0.60	0.60	YES	1.90
day_18	0.20	0.70	0.70	0.70	0.20	0.70	0.70	0.70	0.70	NO	2.00
day_19	0.20	0.70	0.70	0.70	0.20	0.70	0.70	0.70	0.70	NO	2.00
day_20	0.20	0.70	0.70	0.70	0.20	0.70	0.70	0.70	0.70	YES	2.00

Table 3.5: Trend of the third approach for the artificial dataset.

Days	cnt_2xx	cnt_4xx	cnt_5xx	cnt_host	cnt_URL	cnt_payl	avg_payl	un_payl	Reinf. tilt	C
day_1	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	NO	2.00
day_2	0.40	0.50	0.30	0.30	0.40	0.50	0.30	0.50	YES	2.00
day_3	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	NO	2.10
day_4	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	NO	2.10
day_5	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	NO	2.10
day_6	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	NO	2.10
day_7	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	NO	2.10
day_8	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	NO	2.10
day_9	0.30	0.40	0.20	0.20	0.30	0.40	0.20	0.40	YES	2.10
day_10	0.20	0.30	0.10	0.10	0.20	0.30	0.10	0.30	NO	2.20

Table 3.6: Trend of the third approach for the real dataset.

Results will be evaluated and discussed in Chapter 6.

3.5 Program language and supports

Python[27] is used as the programming language for this approach. This language, in fact, is well suited for all those tasks of data analysis, data visualization or that make use of Machine Learning algorithms. Considering also that the dataset used in this approach, which is an example of a real case, is not very large, there were no performance problems in the use of this programming language.

To support Python, the following libraries are used:

- *Pandas*[28], that is a library for data analysis;
- *NumPy*[29], a useful library for using mathematical functions and generating random numbers and probability distributions;
- *matplotlib*[30], a library used for plots creation and visualization in Python.

Chapter 4

Engines for Big Data processing: Spark and GraphX

Big Data processing is a set of programming techniques or models that allows to access large-scale data and extract useful information from it to support services or providing decisions. Typically, in the literature, Big Data processing is defined as any process characterized by the "*Five Vs*":

- the *Volume*, measured in Bytes, which defines the amount of data processed;
- the *Velocity*, which defines the speed with which data is processed;
- the *Variety*, which defines the diversity with which the data has been collected;
- the *Validity*, which denotes the quality of the data considered and its reliability;
- the *Value*, which defines the value and meaning of the data itself in the context considered.

Among Big Data processes, the *Spark* framework definitely stands out, as it has

been very successful in reducing the amount of code needed to create a specific application.

4.1 Apache Spark

Apache Spark[31] is a high performance unified analytics engine for Big Data processing. Spark does not have a data management system and therefore is usually deployed on Hadoop or other storage platforms but can be up to one hundred times faster than a MapReduce paradigm as it is characterized by having an *in-memory cluster computing*, which in fact allows to greatly increase the speed of data processing.

A cluster is a group of nodes (interconnected computing machines) coordinated with each other. Being able to use, therefore, the resources of many processors, a cluster can be very performant and also quite scalable: if it needs to have more resources, then more processing capacity, just introduce more nodes in the cluster.

Spark makes use of dedicated modules for Data Streams, SQL, Machine Learning and Graph Processing.
It provides APIs for Java, Scala, Python and R languages.

4.1.1 Spark execution modes

As mentioned, Spark is composed of a number of nodes working in parallel.
Spark has two application execution modes: *Cluster Mode* and *Client Mode*.

In the Cluster Mode, there is a type of architecture defined as *master-slave*, where the master is represented by the driver node and the slaves by the executors nodes. Inside the driver the main Spark program is executed, the *Driver Program*, in which the *Spark Context* object is defined and which has the role of connecting to the *Cluster Manager*. A Cluster Manager has the task of allocating and distributing resources between applications. Once connected, Spark takes possession

of the executors on the cluster nodes, i.e. the processes that perform calculations and store data in parallel, and sends them the code of the application itself that needs to be executed.

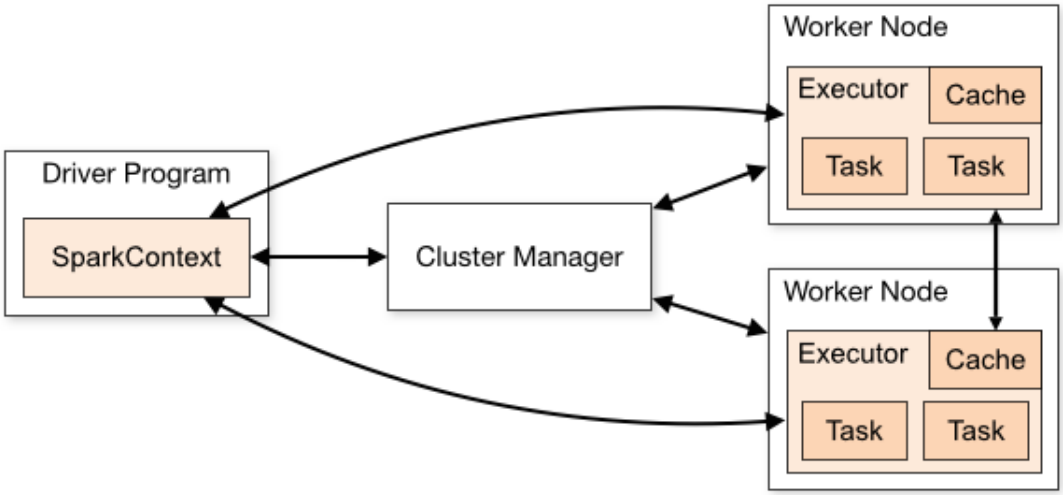


Figure 4.1: Cluster mode overview of Apache Spark.

The executor then, will have two main tasks: to execute the application code that the driver, through the Spark Context, sent to it, and, once executed, to send the result to the driver.

In Client Mode, on the other hand, Driver program and SparkContext run outside the cluster (they will run on the machine from which job is submitted), and is therefore a valid mode for situations where there is no need to use a cluster or to run everything on a single machine.

4.1.2 Resilient Distributed Dataset

Spark is fault tolerant thanks to *Resilient Distributed Datasets (RDD)*.
An RDD is the Spark's main abstraction, which can contain within it any type of object (primitive types or user-defined classes). It represents, therefore, a collection of elements, partitioned among cluster nodes, and it can be processed in parallel.

An RDD can be created from any collection, through SparkContext's *parallelize* function, which takes as parameter an existing collection in the driver program (e.g. a Scala sequence) or from an external dataset, thanks to SparkContext's *textFile* method which takes as parameter the URI of the file and reads it as a collection of lines.

Supporting RDDs are two types of operations: *transformations*, which create a new RDD from an existing one, and *actions*, which return a value from the executor to the driver program after performing a computation on the RDD. Among the transformations, for example, there is the *map*, which passes a function to each element of the RDD and returns a new one with the results obtained. Among the actions, instead, there is the *reduce* that aggregates all the elements of the RDD returning only one value as final result of the operation.

In Spark, transformations are *lazy*, meaning they are not executed until an action needs to compute a value to return to the driver. This behavior improves performance considerably because Spark keeps track of the sequence of transformations that are set up as an execution plan, through a *Directed Acyclic Graph*, to be analyzed and possibly optimized. In addition to this, the improvement is also in Manageability and software complexity.

4.1.3 Spark Components

The main components of Spark are:

- *Spark Core*, which is the core of Apache Spark. The Spark Core provides the execution engine, the internal compute memory, and the reference to datasets stored in external storage systems. It is responsible for executing I/O functions, scheduling, monitoring, memory management, and failures handling.

- *Spark Streaming* which is used to analyze a continuous stream of data. It provides APIs for managing data streams, but ensuring failures handling, throghput and scalability.

- *Spark SQL* which, thanks to the abstraction of objects offered by *Dataframes*, is a support working with data through a SQL-Like approach, in a fast and distributed way.
- *MLib*, that is a machine learning library of Spark that guarantees high speed and high quality algorithms;
- *GraphX*, an engine for graph-parallel computation.

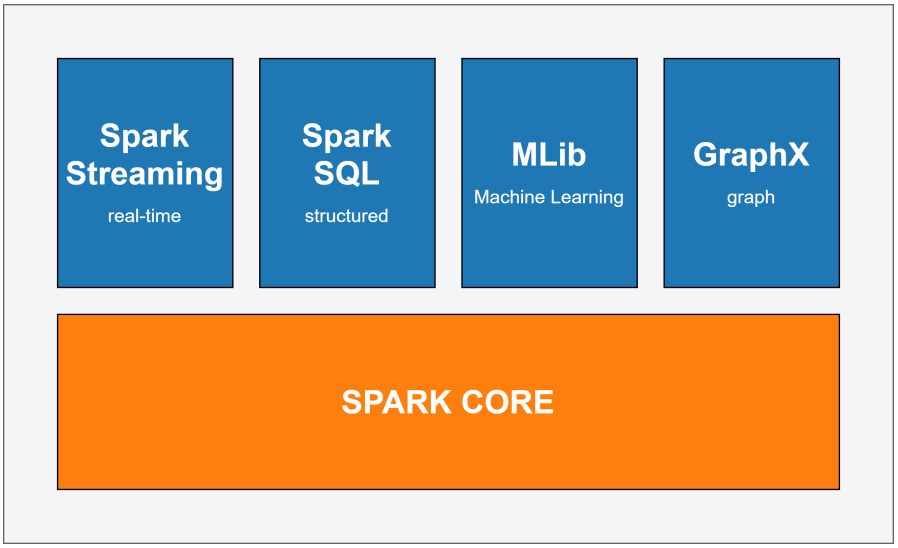


Figure 4.2: Spark Components.

4.2 GraphX

GraphX[32] is a component of Spark, used for creating graphs and performing calculations on them. GraphX, from a functional point of view, extends the concept of RDD already present in Spark with a new level of abstraction that is used to represent a graph by implementing a direct multigraph that allows the presence of attributes for nodes and edges.

In GraphX, a graph is a *Property Graph*.
A Property Graph is a directed multigraph (i.e., a directed graph that allows

multiple edges for each source-destination pair nodes) with user-defined objects paired at vertices and edges, which represent the respective attributes (or properties). Being a multigraph, each source-destination pair nodes can share multiple properties, embedded in the different edges. Each vertex has a vertexId, defined by a 64-bit long number.

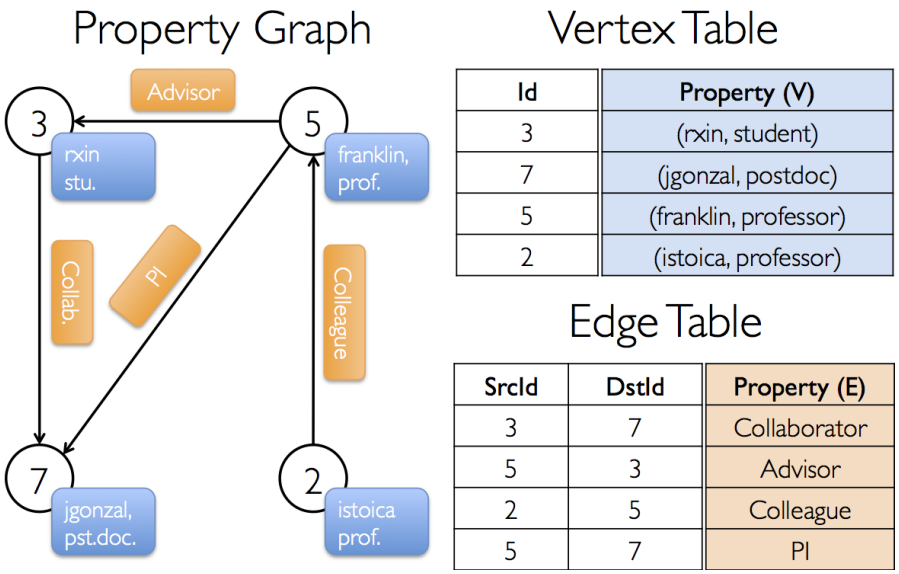


Figure 4.3: GraphX property tables.

GraphX has a large number of operators for processing graph calculations, such as extracting subgraphs or aggregating messages. It also contains a large collection of graph constructors and algorithms for graph analysis. In addition, it provides a variant of the Pregel API, which is one of the key aspects of the algorithm that makes use of graphs developed in this thesis work.

4.3 Pregel API

Pregel is a paradigm for exchanging messages among nodes of a graph and their neighbors. It was first published by Google in 2010[33]. The Apache GraphX library provides an API for computing on graphs using the Pregel paradigm.

The method signature of Pregel, in Spark, is:

```
def pregel[A](
    initialMsg: A,
    maxIter: Int = Int.MaxValue,
    activeDir: EdgeDirection = EdgeDirection.Out)(
    vprog: (VertexId, VD, A) => VD,
    sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
    mergeMsg: (A, A) => A)
    : Graph[VD, ED]
```

where `VD` is the vertex attribute type and `ED` is the edge attribute type. Thanks to the availability of the GraphX Property Graph, vertex and edges attributes can be used within the Pregel paradigm.

As shown, the function accepts two lists of parameters. The first list contains the values:

- `initialMsg` which indicates with which "message" each vertex will start its iteration;
- `maxIter` which represents the upper limit on the number of iterations the computation can have, considering that for several iterations, the message arrived at a node can be sent back to its neighbors;
- `activeDir` which indicates the direction of message propagation, whether from source node to destination node, from destination to source node, or both.

The second list contains user defined functions which are:

- the vertex program `vprog`, which represents how each vertex should be updated after waiting for the result of the mergeMsg function. In this function, therefore, there is the construction of the new attributes for the vertex considered;
- the send messages `sendMsg` that indicates the message that must be sent and to which vertex, in fact the first element of the returned iterator indicates the vertex of the destination node while the second element indicates the

message itself. As shown, vertices or edge attributes can be retrieven by the *EdgeTriplet Class*;

- merge messages `mergeMsg` that indicates how the various messages that a node receives are aggregated (exactly like a reduce function for an RDD).

At the end of the Pregel function run, a new graph is returned.

Among the special features of Pregel there is the possibility to send messages both from the source node to the destination node and vice versa. Furthermore, since vprog, sendMsg and mergeMsg are functions, the algorithm can be made extremely sophisticated and customizable. But the real strength of Pregel is the possibility to exchange any kind of message, including lists, sequences or Tuple objects. In this way, the paradigm is very useful for building very sophisticated graph algorithms.

Chapter 5

Second approach: detecting malicious hosts with a graph-based method

The use of graphs in the cybersecurity world, as also discussed in Chapter 2, is very diverse. In this thesis work, however, the proven approach is based on a *scoring mechanism*. The score, which is an information present for each host towards which users have made a connection, and also present in the users themselves (as will be discussed later), represents a certain degree of "maliciousness".

Starting from hosts whose maliciousness is already known, thus hosts to which users connect only if hacked, the algorithm gives a certain basic score, equal for all. Then, through Pregel's paradigm, for a given day, this score can be first uploaded to users, who will accumulate all the scores of the hosts they connected to, and then, the next day, downloaded to new hosts, not present the previous day, thus being able to understand which of them can be classified as malicious or not.

The algorithm allows the detection of new malicious hosts that have never been seen before within a company's network, or never known as such.

5.1 Input datasets

The datasets used for the graph approach are two: the *artificially created dataset* an the *real one*, discussed in Section 3.1 and already used in the statistical approaches part.

5.2 The algorithm

The algorithm is daily: it reads the data of a certain day taken into consideration and makes the analysis having a certain knowledge of the past days, for several consecutive days.
First, two empty RDDs are created, the `cumulativeVerticesRDD` and the `cumulativeMalHostsRDD`: the first one has the purpose of collecting daily the users who have made at least one connection, in order to keep them available for the analysis of the following days, while the second one has the purpose of collecting, day after day, all the hosts considered malicious by the algorithm.

For each considered day, the `connectionsRDD` is created through the function "textfile" made available by SparkContext passing as parameter the name of the `csv` file of the considered day. The second step is the removal of the header of the file, useless information for the algorithm, and this is done through the `filter` transformation and checking if the string considered is equal to that of the header. At this point, the output RDD, the `cleanConnectionsRDD`, which represents the set of connections of users to hosts, is made available for the next steps.

Considering all the runs of the algorithm, and that the RDD is created once a day, in total the creation of the `cleanConnectionsRDD` is done 50 times for the artificial dataset and 25 times for the real dataset, one RDD for each day.

5.2.1 Vertices mapping

The mapping of the vertices is done through the function `flatMap`, that has the same mechanism as the map function but with the particularity that the RDD returned can have the same number or more elements of the RDD to which it applies. In fact, *two vertices* are created for each connection, the one representing the user and the one representing the host.

Starting with the `cleanConnectionsRDD`, each day all vertices of the graph are created. They are represented by the `verticesWithPointsRDD` and a single vertex is of type `(Long, (String, String, Float, Float))`. This is a `Tuple2` object (an object that contains a fixed number of elements, each with its own type) with a `Long` as the first parameter, representing the *vertexId*, and a `Tuple4` as the second parameter, representing the *attributes* for each vertex.

The first parameters are simply taken from the considered daily dataset, using as userId the id extracted from the `user` field, when considering the artificial dataset, or the `cip` field, when considering the real dataset. For the hostId, instead, the id extracted from the `host` field is used when considering the artificial dataset, while the `dst` field is used when considering the real one, but it is added to 1000000 if it is a "normal" host or 3000000 if it is a "malicious" host. The hostIds can be used as they are provided but this sum is for better display of the results later on, not for the execution of the algorithm itself.

The first two fields of the second parameter, `Tuple4` object, are respectively the *generic identifiers*, used to have a more readable data, i.e. the *user* or *host* fields for the artificial dataset and *cip* or *dst* fields for the real one, and the connection *label*, retrieven directly from the dataset, and which initially is always `OK` for the users while it is then evaluated for the hosts. The second fields, instead, are the *score*, which initially is always 0 for the users, while it is evaluated for the hosts, and an *additional field* that is used for subsequent calculations and initially always set to 0, for both users and hosts.

The creation of the user is basically over since its most important information

has already been processed. As far as the host is concerned, always starting from the `label` field present in the dataset, if it is malicious then the corresponding vertex of the graph has the second field of the `Tuple4` object, which represents the label, set to `MAL` and the third field, which represents the score, equal to 100. If, instead, the host is not malicious, the label of the vertex is `OK` and its score equal to 0. In this way, starting from malicious hosts a priori, it has begun to give a certain score to them, that of base is always equal to 100, for every connection.

On the new obtained RDD is launched a reduceByKey, which merge the values using an associative and commutative reduce function, but for each key of the RDD (in this case the vertexId). In this reduce function there is the sum of all scores accumulated for each vertex, in fact, within the dataset, a user or a host may appear several times, and in the case that the host is malicious, it is fair that it has a higher score than a malicious one that appears much less often. In this way, the algorithm is able to give a certain unbalance on the hosts, just to be able to give a greater weight to all those hosts that appear more often, otherwise everything would be much more flat and static.
Nothing changes, instead, for non-malicious hosts, because they have an initial score always equal to 0.

In Figure 5.1, is shown an example of the vertices created, users and hosts, with their Ids and initial attributes.

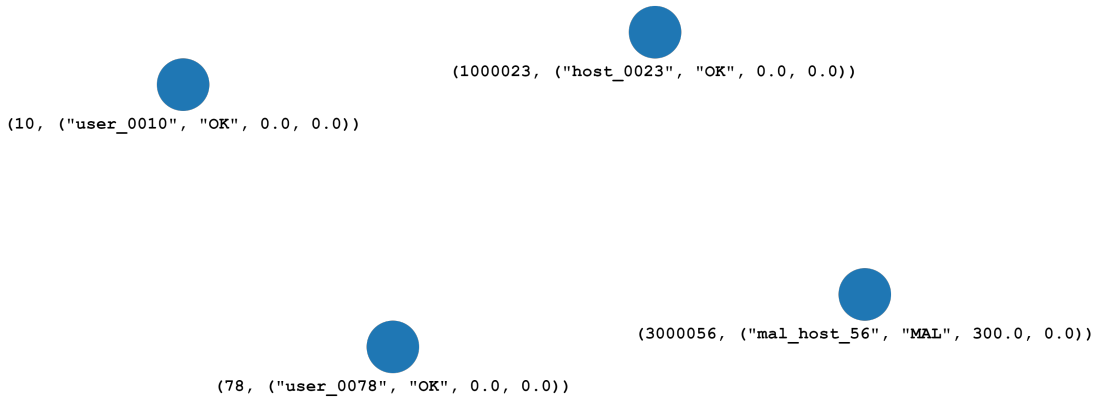


Figure 5.1: Example of a graphic representation of the reducedVerticesWithPointsRDD for the artificial dataset.

At this point, the `reducedVerticesWithPointsRDD`, representing all vertices, is ready for the creation of the graph.

5.2.2 Edges mapping

Each edge of the graph represents a connection of the dataset. An edge is represented by an object of type `Edge(Long srcId, Long dstId, ED attr)`. The first parameter, `srcId`, represents the vertexId of the source node, the user, while the second parameter, `dstId`, represents the vertexId of the destination node, the host. The third parameter `attr` represents the set of attributes for the considered edge. In this case a `Tuple2` object is chosen with the HTTP status of the connection as the first parameter and the payload as the second. All edges informations are extracted from the `cleanConnectionsRDD` using a map transformation.

The user vertexId and the host vertexId, for `srcID` and `dstID` parameters, are extracted in the same way as described in the previous section. For the attributes information, the HTTP status is extracted from the `status` field for the artificial dataset or from the `scstatus` field for the real one, while the payload is extracted from the `payload` field for the artificial dataset or from the `csbytes` field for the real one.

In Figure 5.2, is shown an example of the edges created, with their srcId, dstId

and attributes.

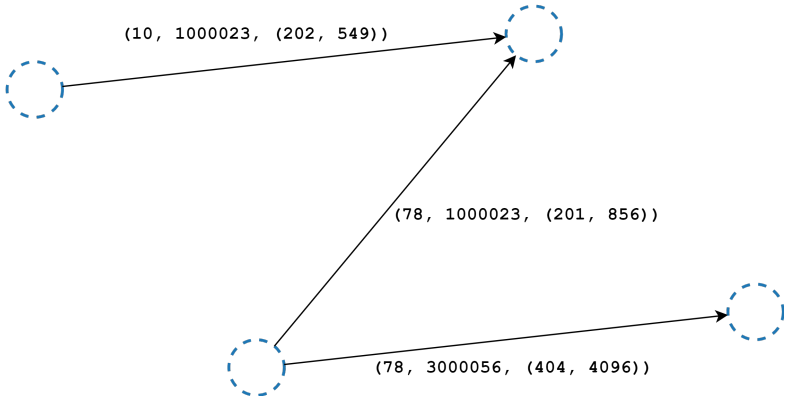


Figure 5.2: Example of a graphic representation of the edgesRDD for the artificial dataset.

At this point, the `edgesRDD` representing all edges is ready for the creation of the graph.

5.2.3 PregelIN

The graph is constructed using the `reducedVerticesWithPointsRDD` for the vertices and the `edgesRDD` for the edges.

The purpose of the first use of the pregel paradigm, called *PregelIN*, is to accumulate scores among users. If a user has had a connection with a malicious host (whose maliciousness is known a priori) he is then entitled to earn his score. Each user, in the end, will then have an accumulated score equal to the sum of the scores of all hosts they have connected to.

The first use of the pregel algorithm takes as parameters two lists, the first of which is composed of:

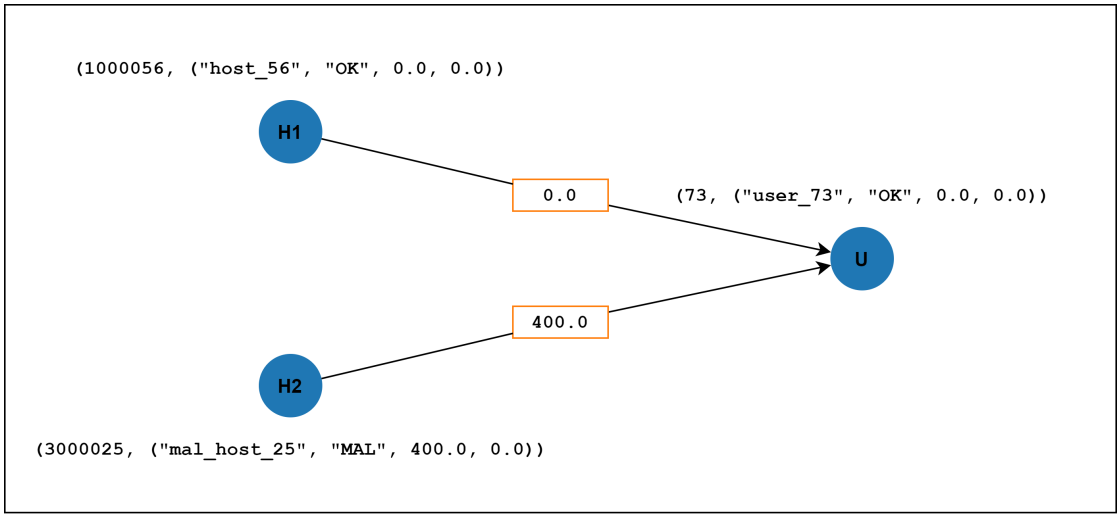
- an `initialMsg = 0.0f`, because the first message of the iteration can be a "default" one;

- a `maxIterations = 1`, since the run of the paradigm happens only once, in fact the messages are distributed only once between the nodes;
- `activeDirection = EdgeDirection.In`, since at this stage messages are sent from destination nodes to source nodes (i.e. from hosts to users).

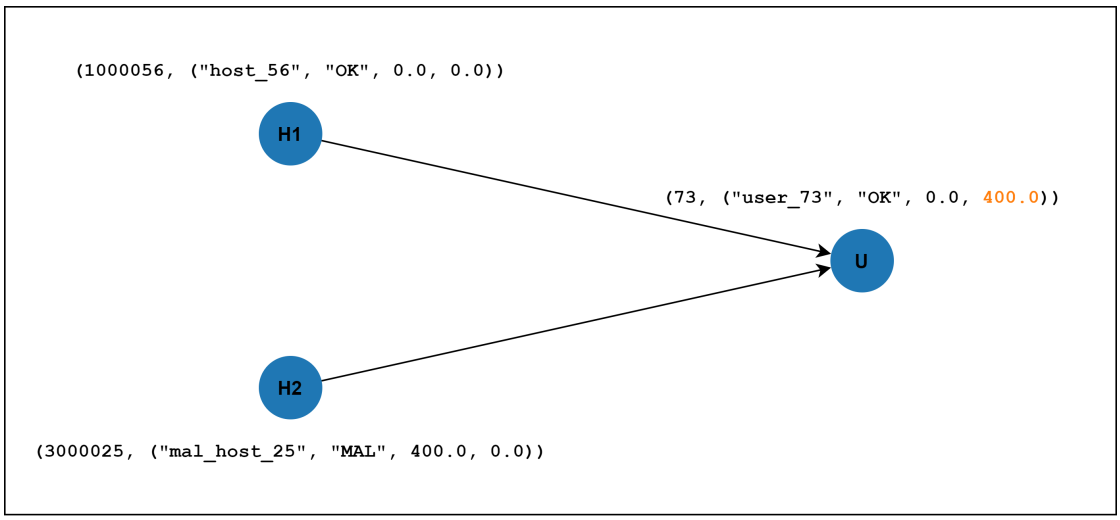
The second list, on the other hand, takes three user defined functions as parameters, which are:

- the vertex program `vprog`, which receives as parameters a `Long` type as the `vertexId`, the attributes of the vertex being considered, composed by a `Tuple4` object of `(String, String, Float, Float)` and, finally, a `Float` number, that is the output of the `mergeMsg`. This function returns a `Tuple4` that has the same values as the vertex being considered except for the fourth element, which is summed with the output of the mergeMsg. In fact, after the aggregation of all the received messages, there is the definition of the new attribute for the considered node, which will be nothing else than the previous one but with the updated value of the accumulated score (in this case the fourth field of the attributes that initially was always 0 for users);
- the computing message function, the `sendMsg`, which takes as parameter an `EdgeTriplet` and returns an `Iterator`. The EdgeTriplet allows to extract information (the attributes) about the source and destination nodes and the edge that connects them. Thus, for the iterator, the `edgeTriplet.srcId` is passed as the first parameter, which indicates that the direction of the message is from the destination node to the source node, and as the second parameter is passed the score of the destination node (the initial score of the host).
- the aggregation messages function `mergeMsg`, which, acting exactly like a reduce function, receives as parameters two `Float` numbers, that are two messages arrived to the node and returns their sum.

The PregelIN mechanism is shown in Figure 5.3.



PregelIN during sendMsg



PregelIN after vprog

Figure 5.3: PregelIN mechanism.

The users of the graph, after the PregelIN phase, are kept available for the PregelOUT one, then they will be concatenated to the vertices found on the next day in order to start the PregelOUT phase. In fact, the hosts considered malicious allowed the accumulation of a score, by the users, that will be used the next day, for the detection of new malicious hosts never seen before. The concatenation is used to aggregate the users of the graph with the vertices of the next day.

5.2.4 PregelOUT

The second use of Pregel, called *PregelOUT*, which makes use of the score accumulated by the pregelIN on the previous day, has the purpose of downloading the score uploaded by users to the hosts connected to that day. In fact, a user who uploaded a certain score on the previous day is considered hacked, because he may have taken points only from malicious hosts (or that the proxy server considered as such) and can download it to the hosts it has connected to, the day considered, because most likely these will be malicious as well. These hosts on which the score is downloaded are certainly hosts not seen by the user the previous day, so this mechanism allows to identify new malicious hosts never seen before by the user, or never known as such.

The mechanism of download, is made even more complicated thanks to the customization of the pregel function. In fact, through an analysis of the considered connection, and therefore of the edge, the downloaded score is not only a single number representing the score that the user has acquired the previous day, but a set of other parameters that are useful in the final classification.

Initially, the `cumulativeVerticesRDD` is mapped in order to have a `Tuple7` as attribute for the nodes. In fact, three new empty fields (all set to 0) are added as they are useful for the subsequent pregel algorithm. The `reducedVerticesWith0PointsRDD` is thus obtained and used for the creation of a new graph with the `edgesRDD`.

The PregelOUT paradigm used on the new graph takes two lists as input parameters. The first list consists of:

- `initialMsg = (0.0f, 0.0f, 0, 0)`, which are the default values to be used on the first iteration;
- `maxIterations = 1`, because the iteration of the paradigm is only once;
- `activeDirection = EdgeDirection.Out`, because the message is sent from source nodes to destination nodes (i.e. from users to hosts).

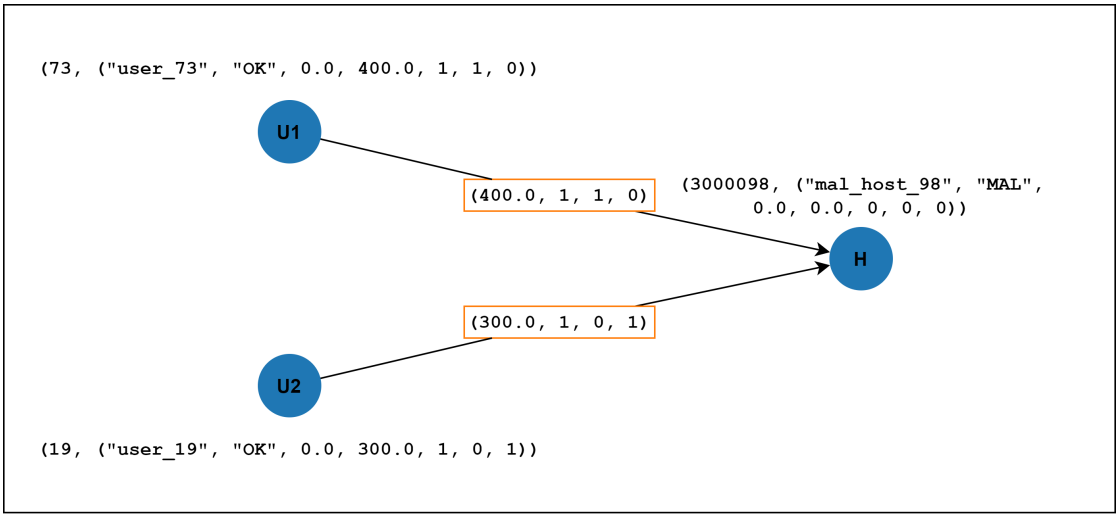
The second list, on the other hand, has the following user defined functions:

- the vertex program `vprog` which receives as parameters a `Long`, representing the vertexId, a `Tuple7`, representing the initial attributes of the considered node, and a `Tuple4`, representing the output of the mergeMsg. The returned value is a `Tuple7` and represents the new set of attributes for the considered node. The first three are the same as the previous ones (i.e. the generic identifier, the initial label and the initial score) while the remaining four are aggregations of the previous attributes with those received as output from the mergeMsg.

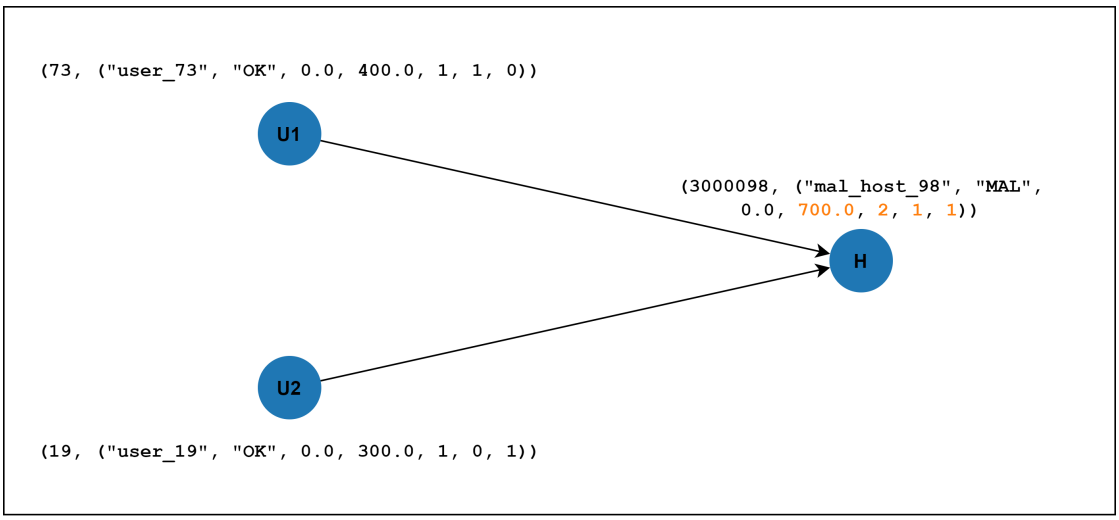
- the computing message function `sendMsg`, which takes an `EdgeTriplet` as a parameter and returns an `Iterator`. Through the EdgeTriplet, within the function, informations are extracted to evaluate even more precisely the connection, i.e. the status and the payload. These two pieces of information are examined to choose whether to set a certain number of variables to 0 or 1. These variables are the `commonPayload`, which is set to 1 if the payload is equal to the `frequentPayload`, the `bigPayload`, which is set to 1 if the payload is larger than the `payloadThreshold` and the `badStatus`, which is set to 1 if the status is not between 200 and 299. The `frequentPayload` and the `payloadThreshold` are values set at the beginning of the program by security operators and based on analysis made on the same network previously. In the iterator, the returned values are the `edgeTriplet.dstId` as the first parameter, to indicate that the message should be sent from the source node to the destination node, and a `Tuple4` object as the second parameter, which encapsulates the user node's score and the three values of `commonPayload`, `bigPayload`, and `badStatus`.

- the message aggregation function `mergeMsg`, which receives as parameters two `Tuple4`, i.e. two messages arrived at the node, and returns a `Tuple4` object containing the sums of the values of the previous two. In this function there is the sum of all scores received, the sum of all `commonPayload`, the sum of all `bigPayload` and, finally, the sum of all `badStatus` received from neighbors nodes.

The PregelOUT mechanism is shown in Figure 5.4.



PregelOUT during sendMsg



PregelOUT after vprog

Figure 5.4: PregelOUT mechanism.

5.2.5 Computation of final score

The experiment continues with the normalization of the results. Among all vertices of the graph, the maximum and minimum values are extracted for attribute points, then for the scores accumulated after the PregelOUT phase, as well as the sums of `commonPayload`, `bigPayload`, and `badStatus` fields.

Considering the formula:

$$Z_n = 100 \cdot \frac{Z - min}{max - min} \tag{5.1}$$

and considering the maximum and minimum values extracted for the four attributes, each vertex is mapped to derive the normal values for each of them, obtaining:

- the normalized score accumulated by the host, as nS;
- the normalized total number of connections with payloads equal to a certain anomalous frequent payload that the host had in the day considered, as nCP;
- the normalized total number of connections with payloads greater than to a certain anomalous big payload had that the host had in the day considered, as nBP;
- the normalized total number of connections with bad status that the host had in the day considered, nBS;

At the end, for each host of the graph, the final score is calculated through a map function. The final score is given by the formula:

$$finalScore = p_1 \cdot S + (1 - p_1) \cdot (p_{21} \cdot nCP + p_{22} \cdot nBP + p_{23} \cdot nBS) \tag{5.2}$$

where

- $0 \leq p_1 \leq 1$;
- $p_{21} + p_{22} + p_{23} = 1$;

The intent of this formula is to get a weighted score for all four characteristics, but still give more weight to the score obtained. The will to use also the three features, derives from the fact to be able to use also the information present inside a connection, and therefore the evaluation of these when anomalous, and not only the information belonging to the single vertices. In fact, any security

operator, who already knows the company network, can modify, according to the needs, the values of p_1, p_{21}, p_{22} and p_{23}, also thanks to an analysis made some time before the considered day, as a normal monitoring activity. He can also change the `frequentPayload` and `payloadThreshold` values, used in the pregelOUT algorithm, knowing which can be anomaly indices for the considered dataset.

Experimentally, the best values found for the weights are:

- $p_1 = 0.8$ for the artificial dataset and $p_1 = 0.9$ for the real one;
- $p_{21} = 0.6$ for the artificial dataset and $p_{21} = 0.3$ for the real one;
- $p_{22} = 0.2$ for the artificial dataset and $p_{22} = 0.3$ for the real one;
- $p_{23} = 0.2$ for the artificial dataset and $p_{23} = 0.4$ for the real one.

The `scoresRDD` is then obtained, having the final score for each host.

5.2.6 Classification

By choosing a *threshold*, the algorithm proceeds with the classification of malicious and non-malicious hosts. The `scoresRDD` is mapped, such that every host's final score is compared to the chosen threshold. If this is greater than the threshold, the host would be labeled with `MAL` denoting its maliciousness, or with `OK` otherwise, thus obtaining the `classificationResultsRDD`.

The value of chosen threshold is a value that only for the first iteration is held static, and chosen equal to 80.0, but subsequently this changes in dynamic way based on the results of the previous days. The `classificationResultsRDD`, in fact, is first filtered in order to obtain only the vertices classified as `MAL`, and then it is mapped with only the values of the scores in order to obtain, at the end, their average. In this way it is obtained the average score for all the hosts classified as `MAL`. This average will be the threshold considered, the next day, for the classification of new hosts. In the same way, but only for statistical information, the procedure is also applied to all hosts classified as `OK`, so as to obtain their average

score.

From this moment on, the hosts classified as malicious on the day in question are extracted from the `classificationResultsRDD`, through a filter transformation. From this new RDD all malicious hosts found in the previous days, contained in the `cumulativeMalHostsRDD`, are subtracted and the `newFoundMalHostsRDD` is created, i.e. the RDD with all hosts classified as malicious but never considered as such in the previous days, which is made available to the security department that will deepen the analysis on them. Once these new hosts have been found, the `newFoundMalHostsRDD` will be merged with the `cumulativeMalHostsRDD`, so that the operation will be repeated in the following days and the set of new malicious hosts will always be up-to-date.

From this point on, the algorithm, for the day in question, is finished, and will begin by restarting the construction of the graph for PregelIN. In this way the run can be done on a daily basis, thus being able to be used for several days.

5.2.7 Results

Only for the purpose of evaluating results, however, even before starting the PregelIN preparatory to the next day, the algorithm also takes care of deriving all the metrics necessary to establish the goodness of the model (in particular the number of True Positives, True Negatives, False Positives and False Negatives) and which will then be discussed in Chapter 6.

However, an evaluation of the model needs to be done. As shown in Figure 5.5, it is possible to see that the algorithm, after the PregelIN phase, managed to accumulate a certain score towards users who made connections towards malicious hosts. The figure, in fact, represents all the users who were hacked in a day taken into consideration, and the hosts towards which they connected. For visual purposes, only malicious hosts are represented. The color of the users, moreover, denotes how much score they have accumulated (the darker then the higher the score) making it almost as if there is a certain "degree of maliciousness", as if one

user is "more infected" than others. In fact, a user infected with a virus that is more powerful than others can be even more dangerous within the company.

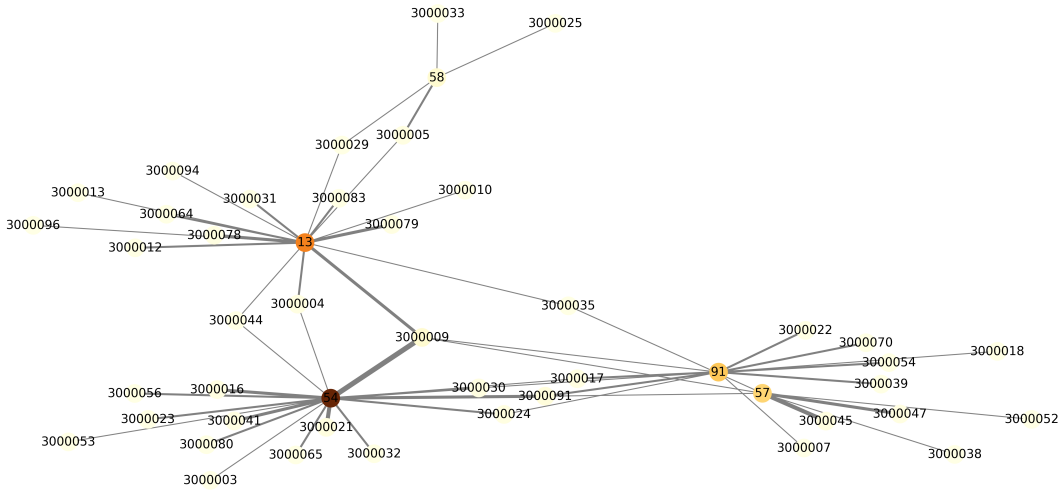


Figure 5.5: PregelIN example.

It is noticed, moreover, that the thickness of the edge between a user and a host, is index of the number of connections had between the two (more it is thick therefore more connections had) denoting like the darker nodes have thicker edges, and therefore that the users with higher score tend more to make connections with malicious hosts than others.

The next day, instead, there is the download of the score, as it appears in Figure 5.6, through the PregelOUT phase, from the users who were hacked the previous day to the nodes they connected to just on the new day. As shown, in fact, there are new all malicious hosts (there are also some false positives but for visualization reasons they have been omitted) but that the previous days had not been known as such.

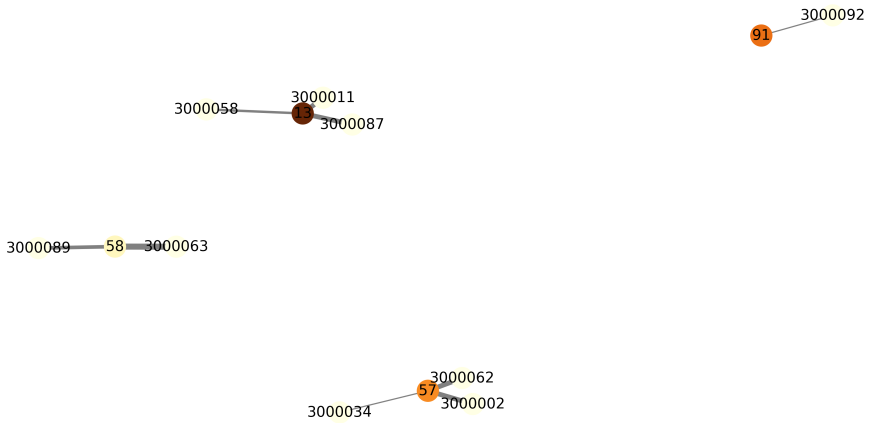


Figure 5.6: PregelOUT example.

5.3 Program language and supports

The algorithm is written in the Scala language. The choice of this language is mainly due to two reasons: the Scala API for GraphX is in a very stable version and because Scala allows easy and clean use of Lambda Expressions, which in this context are used intensively for all Spark transformations and actions. As support, the Log4j library is also used, which provides statement log support, so that the results can be easily and clearly read for model goodness-of-fit analysis.

Chapter 6

Metrics and results

Because the experiments studied in this thesis work were, for both main approaches used, classification tasks, between users considered hacked or not, or between hosts considered as malicious or not, the following metrics for evaluating the results are taken into account.

Given:

- *TP* as the number of *True Positives*, i.e. the number of users correctly classified as hacked for the first approach or the number of hosts correctly classified as malicious for the second one;

- *TN* as the number of *True Negatives*, i.e. the number of users correctly classified as not hacked for the first approach or the number of hosts correctly classified as not malicious for the second one;

- *FP* as the number of *False Positives*, i.e. the number of users incorrectly classified as hacked for the first approach or the number of hosts incorrectly classified as malicious for the second one;

- *FN* as the number of *False Negatives*, i.e. the number of users incorrectly classified as not hacked for the first approach or the number of hosts incorrectly classified as not malicious for the second one;

the Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

The Accuracy represents the ratio of the number of correct predictions to the total number of observations. In this thesis work is not the most widely held metric, since the datasets used are quite unbalanced in the proportionality of the classe labels.

The Precision and the Recall, instead, are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

and

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

The Precision is the ratio of the number of correct positives predictions (hack users or malicious hosts) to the total number of observations predicted as positives, while the Recall is the ration of the number of correct positives predictions to the total number of actually positives.

Finally, the F1-Score is defined as:

$$F1\text{-}Score = 2 \cdot \frac{P \cdot R}{P + R} \tag{6.4}$$

where P is the Precision and R is the Recall.

The F1-Score takes into account both of the previous metrics, in fact it represents their harmonic mean. In this thesis work, it is one of the metrics taken into account the most.

However, a clarification must be made: since these algorithms studied are made

ideally to support any security team within the company, in addition to the use of the metrics explained above, it is also done an analysis on the individual TP, TN, FP and FN numbers.

6.1 Results of statistical approaches

The first analysis of results id done on the three approaches that make use of statistical techniques, such as the Percentile-99, the Z-Score, and the Inter-Quartile Range.

6.1.1 Percentile-99

As shown in Table 6.1, the results are obtained for different days and for different values of the number of anomalies considered simultaneously for classifying a user as hacked.

The first day brought poor results: in fact, the dataset is populated by hacked users only in a progressive way, starting from a very low number of them in the first days.

The Accuracy values are very high, but not very significant if it is considered that the dataset is very unbalanced: the number of hacked users is much lower than the number of non-hacked ones.

Considering the Precision values, it is possible to see that they are very high: this means that the model had very few False Positives. Moreover the values improve with the increase of the number of anomalies held in consideration. The Recall, instead, presents a fluctuating trend but without having too high values: this means that the number of false negatives is quite high, and therefore there are many hacked users that the model was not able to identify.

Days	Acc_3	Prec_3	Rec_3	F1-S_3	Days	Acc_4	Prec_4	Rec_4	F1-S_4
day_1	0.980	0.000	0.000	0.000	**day_1**	0.990	0.000	0.000	0.000
day_2	0.980	0.600	1.000	0.750	**day_2**	0.990	1.000	0.667	0.800
day_3	0.970	0.600	0.750	0.667	**day_3**	0.990	1.000	0.750	0.857
day_4	0.990	1.000	0.833	0.909	**day_4**	0.990	1.000	0.833	0.909
day_5	1.000	1.000	1.000	1.000	**day_5**	1.000	1.000	1.000	1.000
day_6	0.970	0.333	0.500	0.400	**day_6**	0.990	1.000	0.500	0.667
day_7	0.990	0.500	1.000	0.667	**day_7**	0.990	0.500	1.000	0.667
day_8	0.980	1.000	0.600	0.750	**day_8**	0.960	1.000	0.200	0.333
day_9	0.980	1.000	0.600	0.750	**day_9**	0.980	1.000	0.600	0.750
day_10	0.950	0.700	0.778	0.737	**day_10**	0.980	1.000	0.778	0.875
day_11	0.950	1.000	0.545	0.706	**day_11**	0.950	1.000	0.545	0.706
day_12	0.960	0.818	0.818	0.818	**day_12**	0.950	1.000	0.545	0.706
day_13	0.940	0.909	0.667	0.769	**day_13**	0.920	0.889	0.533	0.667
day_14	0.900	0.857	0.400	0.545	**day_14**	0.910	1.000	0.400	0.571
day_15	0.930	1.000	0.682	0.811	**day_15**	0.910	1.000	0.591	0.743
day_16	0.960	0.923	0.800	0.857	**day_16**	0.960	1.000	0.733	0.846
day_17	0.960	1.000	0.667	0.800	**day_17**	0.950	1.000	0.583	0.737
day_18	0.940	0.818	0.692	0.750	**day_18**	0.950	1.000	0.615	0.762
day_19	0.960	0.900	0.750	0.818	**day_19**	0.950	1.000	0.583	0.737
day_20	0.900	0.800	0.308	0.444	**day_20**	0.900	0.800	0.308	0.444
Days	**Acc_5**	**Prec_5**	**Rec_5**	**F1-S_5**	**Days**	**Acc_6**	**Prec_6**	**Rec_6**	**F1-S_6**
day_1	0.990	0.000	0.000	0.000	**day_1**	0.990	0.000	0.000	0.000
day_2	0.990	1.000	0.667	0.800	**day_2**	0.990	1.000	0.667	0.800
day_3	0.980	1.000	0.500	0.667	**day_3**	0.980	1.000	0.500	0.667
day_4	0.990	1.000	0.833	0.909	**day_4**	0.980	1.000	0.667	0.800
day_5	1.000	1.000	1.000	1.000	**day_5**	0.940	0.909	0.667	0.769
day_6	0.990	1.000	0.500	0.667	**day_6**	0.990	1.000	0.500	0.667
day_7	0.990	0.500	1.000	0.667	**day_7**	1.000	1.000	1.000	1.000
day_8	0.960	1.000	0.200	0.333	**day_8**	0.960	1.000	0.200	0.333
day_9	0.960	1.000	0.200	0.333	**day_9**	0.960	1.000	0.200	0.333
day_10	0.940	0.818	0.692	0.750	**day_10**	0.970	1.000	0.667	0.800
day_11	0.930	1.000	0.364	0.533	**day_11**	0.930	1.000	0.364	0.533
day_12	0.930	1.000	0.364	0.533	**day_12**	0.910	1.000	0.182	0.308
day_13	0.920	1.000	0.467	0.636	**day_13**	0.920	0.889	0.533	0.667
day_14	0.880	1.000	0.200	0.333	**day_14**	0.870	1.000	0.133	0.235
day_15	0.920	0.889	0.533	0.667	**day_15**	0.850	1.000	0.318	0.483
day_16	0.920	1.000	0.467	0.636	**day_16**	0.900	1.000	0.333	0.500
day_17	0.910	1.000	0.250	0.400	**day_17**	0.890	1.000	0.083	0.154
day_18	0.930	1.000	0.462	0.632	**day_18**	0.910	1.000	0.308	0.471
day_19	0.920	1.000	0.333	0.500	**day_19**	0.920	1.000	0.333	0.500
day_20	0.890	1.000	0.154	0.267	**day_20**	0.890	1.000	0.154	0.267

Table 6.1: Daily results of Percentile-99 technique on the artificial dataset while considering 3, 4, 5 or 6 anomalies simultaneously for the classification of hacked users.

Finally, for the F1-Score, fairly good values are found, and the best ones are obtained for a relatively small number of anomalies taken into account.

6.1.2 Z-Score

The results of the technique that made use of the Z-Score are shown in table 6.2.

Days	Acc_3	Prec_3	Rec_3	F1-S_3	Days	Acc_4	Prec_4	Rec_4	F1-S_4
day_1	0.990	0.000	0.000	0.000	**day_1**	0.990	0.000	0.000	0.000
day_2	1.000	1.000	1.000	1.000	**day_2**	0.990	1.000	0.667	0.800
day_3	0.990	1.000	0.750	0.857	**day_3**	0.990	1.000	0.750	0.857
day_4	0.990	1.000	0.833	0.909	**day_4**	0.990	1.000	0.833	0.909
day_5	1.000	1.000	1.000	1.000	**day_5**	1.000	1.000	1.000	1.000
day_6	0.980	0.500	0.500	0.500	**day_6**	0.990	1.000	0.500	0.667
day_7	1.000	1.000	1.000	1.000	**day_7**	1.000	1.000	1.000	1.000
day_8	0.970	1.000	0.400	0.571	**day_8**	0.970	1.000	0.400	0.571
day_9	0.960	1.000	0.200	0.333	**day_9**	0.960	1.000	0.200	0.333
day_10	0.980	0.889	0.889	0.889	**day_10**	0.980	0.889	0.889	0.889
day_11	0.950	0.700	0.778	0.737	**day_11**	0.940	1.000	0.455	0.625
day_12	0.980	0.909	0.909	0.909	**day_12**	0.970	1.000	0.727	0.842
day_13	0.940	0.909	0.667	0.769	**day_13**	0.940	0.818	0.692	0.750
day_14	0.930	1.000	0.533	0.696	**day_14**	0.920	1.000	0.467	0.636
day_15	0.930	1.000	0.682	0.811	**day_15**	0.930	1.000	0.682	0.811
day_16	0.970	1.000	0.800	0.889	**day_16**	0.940	0.909	0.667	0.769
day_17	0.950	0.700	0.778	0.737	**day_17**	0.940	1.000	0.500	0.667
day_18	0.950	1.000	0.615	0.762	**day_18**	0.940	1.000	0.538	0.700
day_19	0.960	1.000	0.667	0.800	**day_19**	0.950	1.000	0.583	0.737
day_20	0.910	0.833	0.385	0.526	**day_20**	0.900	1.000	0.231	0.375
Days	**Acc_5**	**Prec_5**	**Rec_5**	**F1-S_5**	**Days**	**Acc_6**	**Prec_6**	**Rec_6**	**F1-S_6**
day_1	0.990	0.000	0.000	0.000	**day_1**	0.990	0.000	0.000	0.000
day_2	0.990	1.000	0.667	0.800	**day_2**	0.980	1.000	0.333	0.500
day_3	0.970	1.000	0.250	0.400	**day_3**	0.970	1.000	0.250	0.400
day_4	0.990	1.000	0.833	0.909	**day_4**	0.990	1.000	0.833	0.909
day_5	0.980	0.889	0.889	0.889	**day_5**	0.940	0.909	0.667	0.769
day_6	0.990	1.000	0.500	0.667	**day_6**	0.990	1.000	0.500	0.667
day_7	0.980	0.909	0.909	0.909	**day_7**	0.970	0.929	0.867	0.897
day_8	0.960	1.000	0.200	0.333	**day_8**	0.960	1.000	0.200	0.333
day_9	0.960	1.000	0.200	0.333	**day_9**	0.950	0.000	0.000	0.000
day_10	0.960	1.000	0.556	0.714	**day_10**	0.940	0.818	0.692	0.750
day_11	0.940	1.000	0.455	0.625	**day_11**	0.930	1.000	0.364	0.533
day_12	0.950	1.000	0.545	0.706	**day_12**	0.920	1.000	0.273	0.429
day_13	0.900	1.000	0.333	0.500	**day_13**	0.900	1.000	0.333	0.500
day_14	0.900	1.000	0.333	0.500	**day_14**	0.870	1.000	0.133	0.235
day_15	0.880	1.000	0.455	0.625	**day_15**	0.830	1.000	0.227	0.370
day_16	0.920	1.000	0.467	0.636	**day_16**	0.900	1.000	0.333	0.500
day_17	0.910	0.833	0.385	0.526	**day_17**	0.890	1.000	0.083	0.154
day_18	0.920	1.000	0.385	0.556	**day_18**	0.910	1.000	0.308	0.471
day_19	0.940	1.000	0.500	0.667	**day_19**	0.920	1.000	0.333	0.500
day_20	0.900	1.000	0.231	0.375	**day_20**	0.890	1.000	0.154	0.267

Table 6.2: Daily results of Z-Score technique on the artificial dataset while considering 3, 4, 5 or 6 anomalies simultaneously for the classification of hacked users.

The trends of the metrics do not differ much from those of Percentile-99: high values of Accuracy and Precision (but without considering the first day), an improvement of values of Recall, which however some days are quite high, and good values of F1-Score, which once again is probably the best metric to consider, are

found. Note, in fact, how the best performance is for a number of anomalies considered simultaneously equal to 3 or 4.

Although minimally, the trends improved over the Percentile-99 technique, probably because anomaly detection by the Z-Score, for this dataset, works better.

6.1.3 Inter-Quartile Range

Finally, the results of the approach using the Inter-Quartile Range technique, as shown in Table 6.3, are reported.

It is immediately noticeable that there are very high values of both Accuracy and Precision: as already described, the number of hacked users is very low, so it is very difficult to have a high number of false positives, and these two values are high accordingly.

The Recall and F1-Score values, on the other hand, are quite good: they are significantly improved with respect to the previous approaches (note how the values are higher especially with the passage of days, compared to the two previous techniques).

In general, in all approaches, the performance of Accuracy and Precision was remarkably good: this is also due to the not excessive size of the dataset considered and to the little presence of hacked users (which however remains a very realistic case within a corporate network). The trends of the Recall, however, have grown technique after technique, demonstrating how at the end the technique of the Inter-Quartile Range is the one that defines the lowest number of false negatives, which is remarkable considering the greater "importance" of a positive (which is a hacked user) compared to a negative (which is a non-hacked user).

All these values lead to the consideration that the best technique among those tested is the Inter-Quartile Range. The IQR limits for which an anomaly is considered, are, more than the other techniques, able to make a correct detection. It

Days	Acc_3	Prec_3	Rec_3	F1-S_3	Days	Acc_4	Prec_4	Rec_4	F1-S_4
day_1	0.990	0.000	0.000	0.000	**day_1**	0.990	0.000	0.000	0.000
day_2	0.980	1.000	0.333	0.500	**day_2**	0.990	1.000	0.667	0.800
day_3	0.980	1.000	0.500	0.667	**day_3**	0.990	1.000	0.750	0.857
day_4	0.990	1.000	0.833	0.909	**day_4**	0.990	1.000	0.833	0.909
day_5	0.970	0.929	0.867	0.897	**day_5**	1.000	1.000	1.000	1.000
day_6	0.990	1.000	0.500	0.667	**day_6**	0.980	0.889	0.889	0.889
day_7	0.970	0.929	0.867	0.897	**day_7**	1.000	1.000	1.000	1.000
day_8	0.970	1.000	0.400	0.571	**day_8**	0.970	1.000	0.400	0.571
day_9	0.960	1.000	0.200	0.333	**day_9**	0.960	1.000	0.200	0.333
day_10	0.970	1.000	0.667	0.800	**day_10**	0.990	1.000	0.889	0.941
day_11	0.970	1.000	0.727	0.842	**day_11**	0.970	1.000	0.727	0.842
day_12	0.980	0.933	0.933	0.933	**day_12**	0.980	0.889	0.889	0.889
day_13	0.950	1.000	0.667	0.800	**day_13**	0.980	1.000	0.867	0.929
day_14	0.920	1.000	0.467	0.636	**day_14**	0.960	1.000	0.733	0.846
day_15	0.910	1.000	0.591	0.743	**day_15**	0.950	1.000	0.773	0.872
day_16	0.960	1.000	0.733	0.846	**day_16**	0.970	1.000	0.800	0.889
day_17	0.960	1.000	0.667	0.800	**day_17**	0.980	1.000	0.833	0.909
day_18	0.940	1.000	0.538	0.700	**day_18**	0.970	1.000	0.769	0.870
day_19	0.970	1.000	0.750	0.857	**day_19**	0.980	0.933	0.933	0.933
day_20	0.910	1.000	0.308	0.471	**day_20**	0.970	1.000	0.769	0.870
Days	**Acc_5**	**Prec_5**	**Rec_5**	**F1-S_5**	**Days**	**Acc_6**	**Prec_6**	**Rec_6**	**F1-S_6**
day_1	0.990	0.000	0.000	0.000	**day_1**	0.990	0.000	0.000	0.000
day_2	0.990	1.000	0.667	0.800	**day_2**	0.980	1.000	0.333	0.500
day_3	0.990	1.000	0.750	0.857	**day_3**	0.980	1.000	0.500	0.667
day_4	0.990	1.000	0.833	0.909	**day_4**	0.990	1.000	0.833	0.909
day_5	1.000	1.000	1.000	1.000	**day_5**	1.000	1.000	1.000	1.000
day_6	0.980	0.933	0.933	0.933	**day_6**	0.990	1.000	0.500	0.667
day_7	1.000	1.000	1.000	1.000	**day_7**	1.000	1.000	1.000	1.000
day_8	0.980	1.000	0.600	0.750	**day_8**	0.960	1.000	0.200	0.333
day_9	0.970	1.000	0.400	0.571	**day_9**	0.950	0.000	0.000	0.000
day_10	0.980	0.889	0.889	0.889	**day_10**	0.970	0.917	0.846	0.880
day_11	0.980	1.000	0.818	0.900	**day_11**	0.980	0.889	0.889	0.889
day_12	0.990	0.917	1.000	0.957	**day_12**	0.950	1.000	0.545	0.706
day_13	0.970	0.929	0.867	0.897	**day_13**	0.920	1.000	0.467	0.636
day_14	0.970	1.000	0.800	0.889	**day_14**	0.890	1.000	0.267	0.421
day_15	0.990	1.000	0.955	0.977	**day_15**	0.860	1.000	0.364	0.533
day_16	0.980	0.933	0.933	0.933	**day_16**	0.940	1.000	0.600	0.750
day_17	0.990	1.000	0.917	0.957	**day_17**	0.950	1.000	0.583	0.737
day_18	0.970	0.917	0.846	0.880	**day_18**	0.990	0.917	1.000	0.957
day_19	0.990	1.000	0.917	0.957	**day_19**	0.960	1.000	0.667	0.800
day_20	0.960	0.846	0.846	0.846	**day_20**	0.910	1.000	0.308	0.471

Table 6.3: Daily results of Inter-Quartile Range technique on the artificial dataset while considering 3, 4, 5 or 6 anomalies simultaneously for the classification of hacked users.

is also worth noting that the best value of anomalies found on the same day, to consider a user as hacked, is 5, since it has the best F1-Score trend.

6.2 Results of Reinforcement Learning approaches

As described, the results of the three Reinforcement Learning approaches used are now reported. In general, as before, the values of the previous metrics used are evaluated, but the analysis is also done based on when the reinforcement did or

did not occur. Thus, for a given day considered, if the reinforcement is tilted, then the approach has triggered it, so that the results for the next day would change.

6.2.1 First approach of Reinforcement Learning

The approach is described in section 3.4.1 and the results are reported in Table 6.4 and Table 6.5.

Days	Accuracy	Precision	Recall	F1-Score	Reinf. tilt	C
day_1	0.99	0.00	0.00	0.00	NO	2.00
day_2	0.98	1.00	0.33	0.50	YES	2.00
day_3	0.98	1.00	0.50	0.67	NO	1.90
day_4	0.99	1.00	0.83	0.91	NO	1.90
day_5	1.00	1.00	1.00	1.00	NO	1.90
day_6	0.99	1.00	0.50	0.67	YES	1.90
day_7	1.00	1.00	1.00	1.00	NO	1.80
day_8	0.97	1.00	0.40	0.57	YES	1.80
day_9	0.95	0.50	0.20	0.29	YES	1.70
day_10	0.97	0.88	0.78	0.82	NO	1.60
day_11	0.96	0.89	0.73	0.80	YES	1.60
day_12	1.00	1.00	1.00	1.00	NO	1.50
day_13	0.95	1.00	0.67	0.80	YES	1.50
day_14	0.91	0.88	0.47	0.61	YES	1.40
day_15	0.94	0.94	0.77	0.85	NO	1.30
day_16	0.98	1.00	0.87	0.93	NO	1.30
day_17	0.97	1.00	0.75	0.86	YES	1.30
day_18	0.95	0.83	0.77	0.80	YES	1.20
day_19	0.96	0.83	0.83	0.83	NO	1.10
day_20	0.93	0.80	0.62	0.70	YES	1.10

Table 6.4: Daily results of the first approach of Reinforcement Learning on the artificial dataset.

For the artificial dataset, the results are slightly worse than the approach not using reinforcement. Reinforcement tends to work well: at the time it is triggered, there are better results in terms of Accuracy and F1-Score for the day immediately following. Note how the trend in C, however, is always decreasing. This is because each time reinforcement is triggered, the number of false negatives exceeded the number of false positives, and so it tries to be a little less "stringent", thus reducing the C parameter. The solution, however, has not led improvements in the results.

Days	TP	TN	FP	FN	Accuracy	Precision	Recall	F1-Score	Reinf. tilt	C
day_1	48	23359	393	39	0.98	0.11	0.55	0.18	NO	2.00
day_2	214	21063	2041	521	0.89	0.09	0.29	0.14	YES	2.00
day_3	152	21265	1901	521	0.90	0.07	0.23	0.11	NO	2.10
day_4	159	21366	1842	472	0.90	0.08	0.25	0.12	NO	2.10
day_5	100	21947	1430	362	0.92	0.07	0.22	0.10	NO	2.10
day_6	107	22100	1253	379	0.93	0.08	0.22	0.12	NO	2.10
day_7	11	23572	211	45	0.99	0.05	0.20	0.08	NO	2.10
day_8	6	23645	150	38	0.99	0.04	0.14	0.06	NO	2.10
day_9	106	21806	1569	358	0.92	0.06	0.23	0.10	YES	2.10
day_10	116	21482	1831	410	0.91	0.06	0.22	0.09	YES	2.20

Table 6.5: Daily results of the first approach of Reinforcement Learning on the real dataset.

For the real dataset, on the other hand, discrete results are found although not as good as the previous ones. The reinforcement is tilted only three times and the value of the parameter C is always increasing, this means that in all three cases of reinforcement there was a majority of FPs compared to FNs. The algorithm manages to classify several hacked users as such but unfortunately returns a high number of False Positives, even if the number of TPs is not very far from the number of FNs.

6.2.2 Second approach of Reinforcement Learning

The second approach is described in section 3.4.2 and the results are reported in Table 6.6 and Table 6.7.

Note how the approach, has given virtually identical results to the previous approach except for the last three days for the artificial dataset and the third and last day for the real one. This means that, given the mechanism chosen and described, the anomalies are all caused by "too high" values compared to the user's normality, and not by "too low" ones. Despite the few improvements in the results, this still shows that the choice to take into account only a certain type of anomalies rather than others, is reasonable.

Days	Accuracy	Precision	Recall	F1-Score	Reinf. tilt	C
day_1	0.99	0.00	0.00	0.00	NO	2.00
day_2	0.98	1.00	0.33	0.50	YES	2.00
day_3	0.98	1.00	0.50	0.67	NO	1.90
day_4	0.99	1.00	0.83	0.91	NO	1.90
day_5	1.00	1.00	1.00	1.00	NO	1.90
day_6	0.99	1.00	0.50	0.67	YES	1.90
day_7	1.00	1.00	1.00	1.00	NO	1.80
day_8	0.97	1.00	0.40	0.57	YES	1.80
day_9	0.95	0.50	0.20	0.29	YES	1.70
day_10	0.97	0.88	0.78	0.82	NO	1.60
day_11	0.96	0.89	0.73	0.80	YES	1.60
day_12	1.00	1.00	1.00	1.00	NO	1.50
day_13	0.95	1.00	0.67	0.80	YES	1.50
day_14	0.92	1.00	0.47	0.64	YES	1.40
day_15	0.95	1.00	0.77	0.87	NO	1.30
day_16	0.98	1.00	0.87	0.93	NO	1.30
day_17	0.97	1.00	0.75	0.86	YES	1.30
day_18	0.97	1.00	0.77	0.87	NO	1.20
day_19	0.98	1.00	0.83	0.91	NO	1.20
day_20	0.93	1.00	0.46	0.63	YES	1.20

Table 6.6: Daily results of the second approach of Reinforcement Learning on the artificial dataset.

Days	TP	TN	FP	FN	Accuracy	Precision	Recall	F1-Score	Reinf. tilt	C
day_1	48	23359	393	39	0.98	0.11	0.55	0.18	NO	2.00
day_2	214	21063	2041	521	0.89	0.09	0.29	0.14	YES	2.00
day_3	152	21266	1900	521	0.90	0.07	0.23	0.11	NO	2.10
day_4	159	21366	1842	472	0.90	0.08	0.25	0.12	NO	2.10
day_5	100	21948	1429	362	0.92	0.07	0.22	0.10	NO	2.10
day_6	107	22102	1251	379	0.93	0.08	0.22	0.12	NO	2.10
day_7	11	23573	210	45	0.99	0.05	0.20	0.08	NO	2.10
day_8	6	23645	150	38	0.99	0.04	0.14	0.06	NO	2.10
day_9	106	21806	1569	358	0.92	0.06	0.23	0.10	YES	2.10
day_10	116	21486	1827	410	0.91	0.06	0.22	0.09	YES	2.20

Table 6.7: Daily results of the second approach of Reinforcement Learning on the real dataset.

6.3 Third approach of Reinforcement Learning

The third approach, finally, is described in section 3.4.3 and the results for the artificial dataset are reported in Table 6.8.

This is definitely the approach that has brought the best results. It is noticed, for first thing, like the reinforcements are diminished: the system to weights with dynamic change (whose course is visible in the table), in fact, have allowed

Days	Accuracy	Precision	Recall	F1-Score	Reinf. tilt	C
day_1	0.99	0.00	0.00	0.00	NO	2.00
day_2	0.99	0.75	1.00	0.86	NO	2.00
day_3	0.99	1.00	0.75	0.86	NO	2.00
day_4	0.99	0.86	1.00	0.92	NO	2.00
day_5	0.99	0.80	1.00	0.89	NO	2.00
day_6	0.99	1.00	0.50	0.67	NO	2.00
day_7	1.00	1.00	1.00	1.00	NO	2.00
day_8	0.95	0.50	0.40	0.44	YES	2.00
day_9	0.95	0.50	0.40	0.44	NO	1.90
day_10	0.98	0.89	0.89	0.89	NO	1.90
day_11	0.97	0.90	0.82	0.86	YES	1.90
day_12	0.98	0.85	1.00	0.92	NO	1.80
day_13	0.97	0.83	1.00	0.91	YES	1.80
day_14	0.96	0.92	0.80	0.86	YES	1.90
day_15	0.96	0.88	0.95	0.91	NO	1.80
day_16	1.00	1.00	1.00	1.00	NO	1.80
day_17	0.99	0.92	1.00	0.96	YES	1.80
day_18	0.98	1.00	0.85	0.92	YES	1.90
day_19	0.98	0.86	1.00	0.92	NO	1.80
day_20	0.95	0.79	0.85	0.81	YES	1.80

Table 6.8: Daily results of the third approach of Reinforcement Learning on the artificial dataset.

a smaller demand for reinforcements. As a consequence, at the last day, C has reached a value equal to 1.80.

The demonstration of the goodness of the model for the artificial dataset is in the results: these are very high for Accuracy, more realistic for Precision, which has much less values equal to 1, and much higher for Recall (and therefore also for F1-Score). This means that compared to its predecessors, the approach is able to significantly decrease the number of False Negatives, thus succeeding in classifying more hacked users as such.

The trend of the weights, moreover, as previously shown in Table 3.5, outlined a sort of "importance" among the features considered. The anomalies found on the features with higher weights, in fact, are those that have greater relevance in the classification of a hacked user.

For the real dataset, on the other hand, the trend is shown in Table 3.6 and the

results are shown in Table 6.9. There are not very high values of Precision, Recall and F1-Score, compared to the experiment on the artificial dataset, but these are roughly similar to the previous two approaches on the real one. Furthermore, there are better values of Accuracy and the number of False Positives has decreased in an important way. Considering, in a real case, the support of a security department inside the company, which will better investigate the all positives classified by the algorithm, a decrease of False Positives will surely be useful. Moreover, comparing the number of True Positives and False Negatives, which together indicate the total number of real positives, it is possible to see that these two values are not so far apart, thus considering the algorithm as fairly good. Probably the considered days are few, so that the weights have all had a decreasing trend, but, likely, with more days available, these could better denote the most relevant features for this dataset, for this classification purpose.

Days	TP	TN	FP	FN	Accuracy	Precision	Recall	F1-Score	Reinf. tilt	C
day_1	49	23324	428	38	0.98	0.10	0.56	0.17	NO	2.00
day_2	257	20846	2258	478	0.89	0.10	0.35	0.16	YES	2.00
day_3	123	21456	1710	550	0.91	0.07	0.18	0.10	NO	2.10
day_4	144	21540	1668	487	0.91	0.08	0.23	0.12	NO	2.10
day_5	91	22066	1311	371	0.93	0.06	0.20	0.10	NO	2.10
day_6	95	22203	1150	391	0.94	0.08	0.20	0.11	NO	2.10
day_7	9	23586	197	47	0.99	0.04	0.16	0.07	NO	2.10
day_8	5	23657	138	39	0.99	0.03	0.11	0.05	NO	2.10
day_9	86	21955	1420	378	0.92	0.06	0.19	0.09	YES	2.10
day_10	20	23098	215	506	0.97	0.09	0.04	0.05	NO	2.20

Table 6.9: Daily results of the third approach of Reinforcement Learning on the real dataset.

For all three reinforcement learning approaches, the graphs in Figure 6.1 shows the trends in Accuracy, Precision, Recall, and F1-Score for the artificial dataset. The best Accuracy and F1-Score performances are in the third approach. Although all of them are brought to very similar values, it is the third approach that is able to classify the dataset better. Note also how the first and second approaches have similar, or overlapping, trends on some days. As mentioned before, in fact, some features found anomalies only for very high values, so the trend remained almost the same for both cases.

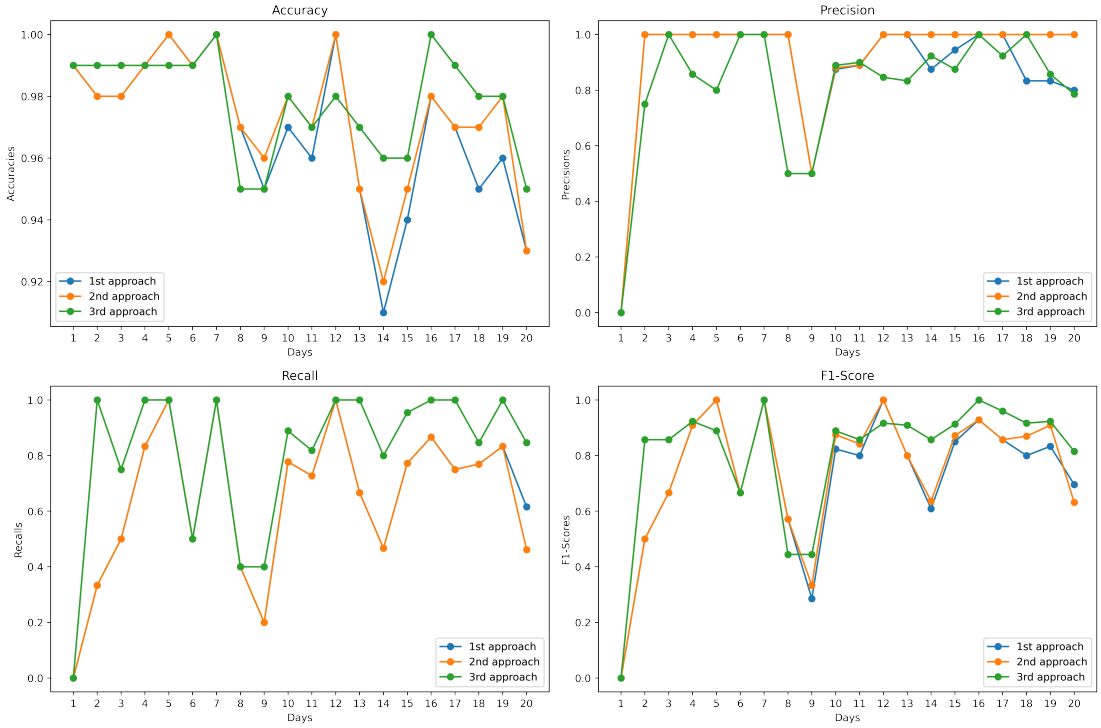


Figure 6.1: Trend of results of the three approaches on the artificial dataset.

Thus, in conclusion, the best results, both for the artificial and the real dataset, are obtained with the third approach that makes use of *weights*. This because it succeeds to distinguish in dynamic way which feature to consider more regarding the others, day by day, adapting itself to the considered dataset and to the different behaviors of the hacked users.

6.4 Results of graph approaches

The results of the graph approach, discussed in Chapter 5, are given below. In addition to the metrics discussed at the beginning of this chapter, individual values of TP, TN, FP, and FN are considered.

6.4.1 Results on artificial dataset

The results of the graph approach tested on the artificial dataset are shown in Table 6.10.

Days	Threshold	avg_OK	avg_MAL	TP	TN	FP	FN	Acc	Prec	Rec	F1-S
day_1	3.20	2.94	3.44	8	1231	472	10	0.72	0.02	0.44	0.03
day_2	3.44	3.46	25.67	35	1234	459	3	0.73	0.07	0.92	0.13
day_3	25.67	4.11	20.49	17	1642	57	40	0.94	0.23	0.30	0.26
day_4	20.49	4.34	19.60	21	1611	84	33	0.93	0.20	0.39	0.26
day_5	19.60	3.74	31.69	20	1654	51	3	0.97	0.28	0.87	0.43
day_6	31.69	2.81	35.77	9	1665	17	10	0.98	0.35	0.47	0.40
day_7	35.77	2.58	21.41	6	1704	8	28	0.98	0.43	0.18	0.25
day_8	21.41	3.07	1.81	0	1683	37	30	0.96	0.00	0.00	0.00
day_9	1.81	3.62	21.53	59	885	803	7	0.54	0.07	0.89	0.13
day_10	21.53	2.69	19.15	27	1667	30	55	0.95	0.47	0.33	0.39
day_11	19.15	3.28	20.72	33	1661	38	53	0.95	0.46	0.38	0.42
day_12	20.72	3.46	21.42	40	1626	52	49	0.94	0.43	0.45	0.44
day_13	21.42	5.11	26.04	45	1612	100	42	0.92	0.31	0.52	0.39
day_14	26.04	4.92	32.57	44	1648	66	48	0.94	0.40	0.48	0.44
day_15	32.57	5.63	27.79	26	1660	38	57	0.95	0.41	0.31	0.35
day_16	27.79	6.02	24.67	26	1642	77	51	0.93	0.25	0.34	0.29
day_17	24.67	5.03	18.32	22	1636	68	49	0.93	0.24	0.31	0.27
day_18	18.32	5.39	18.15	35	1579	136	45	0.90	0.20	0.44	0.28
day_19	18.15	6.04	22.50	40	1542	154	43	0.89	0.21	0.48	0.29
day_20	22.50	5.27	26.41	34	1615	89	36	0.93	0.28	0.49	0.35

Table 6.10: Results of graph approach on the artificial dataset.

It is possible to immediately notice the comparison between the average score of malicious hosts and the average score of non-malicious hosts. The former is always higher than the latter, and this means that hosts classified as malicious always have higher scores than others. The main problem with this trend is that it is never increasing, leading to a worsening of the classifications (remembering that the average score of malicious hosts is chosen as the threshold for the next day).

The classification brought very high accuracy values, and note how this is almost always higher than 0.9. Again, however, one must consider the significant unbalance in the distribution of classes, so it is more appropriate to consider the other metrics. As can be seen, the Precision, Recall and F1-Score values are good. However, when analyzing the number of TPs and FNs in more detail, it can be seen that they are similar, and even often the number of TPs exceeds the number of FNs. In a context of detection in the security field, this can already be considered an excellent result: the presence of malicious hosts is very rare (and therefore

in some ways even more difficult to analyze), and having a certain similarity in the number of TPs and FNs gives great confidence in the goodness of the model.

Moreover, considering the presence of the security department inside the company, if the hosts classified as malicious need further analysis by it, then it possible also to consider as definitely good the number of FPs, related to the number of TPs. For a security operator, the analysis of about one host out of two would lead to the conclusion that the host is really malicious.

6.4.2 Results on real dataset

The results of the graph approach tested on the real dataset are shown in Table 6.11.

Days	Threshold	avg_OK	avg_MAL	TP	TN	FP	FN	Acc	Prec	Rec	F1-S
day_1	80.00	0.12	1.00	0	20180	2	113	0.99	0.00	0.00	0.00
day_2	1.00	0.04	0.46	5	20556	146	103	0.99	0.03	0.05	0.04
day_3	0.46	0.11	1.87	10	19311	762	76	0.96	0.01	0.12	0.02
day_4	1.87	0.09	1.13	4	18852	209	83	0.98	0.02	0.05	0.03
day_5	1.13	0.25	4.81	8	13404	390	24	0.97	0.02	0.25	0.04
day_6	4.81	0.37	3.72	5	6742	135	24	0.98	0.04	0.17	0.06
day_7	3.72	0.18	0.87	3	17253	242	95	0.98	0.01	0.03	0.02
day_8	0.87	0.10	1.59	11	19434	450	91	0.97	0.02	0.11	0.04
day_9	1.59	0.05	0.67	6	19531	125	86	0.99	0.05	0.07	0.05
day_10	0.67	0.05	1.08	9	17719	300	58	0.98	0.03	0.13	0.05
day_11	1.08	0.10	2.88	9	16832	300	47	0.98	0.03	0.16	0.05
day_12	2.88	0.14	2.79	3	11230	124	17	0.99	0.02	0.15	0.04
day_13	2.79	0.34	1.91	3	4120	169	12	0.96	0.02	0.20	0.03
day_14	1.91	0.08	0.40	4	13494	150	56	0.98	0.03	0.07	0.04
day_15	0.40	0.14	1.92	12	17031	731	62	0.96	0.02	0.16	0.03

Table 6.11: Results of graph approach on the real dataset.

As can be seen, although less than in the artificial dataset, also here the trend of the average score among all hosts classified as malicious is always higher than the average score among all hosts classified as non-malicious. But the problem of having an erratic pattern comes up again, which can lead to inaccurate threshold values for the immediate following day.

Accuracies are very high, but since this is also a highly unbalanced dataset in class distribution, it is more appropriate to consider the other metrics. The values of Recall are higher than those of Precision, this means, as also demonstrated by

the columns of FPs and FNs, that the values of FNs are much less than those of FPs. Considering also the values of TPs, these are slightly lower than the values of FNs, it is possible to assume, as already done for the previous dataset, that the approach used is good. In fact, among all the malicious hosts present, a good part of them is found by the algorithm.

The ratio between TPs and FPs is worse than in the previous case, but considering anyway the presence of a security department that will be able to deepen the analysis on all the hosts classified as malicious, the result is then remarkable.

6.5 Performance

In terms of execution time, statistical approaches returned results in times in the order of a few minutes (6-7). In fact, it must be considered that the dataset used is only an example of a real dataset, and therefore the algorithms, even if not used on a cluster of nodes, require little time to be executed. For the real dataset, the execution times were about 6 hours, for each reinforcement approach.

Graph algorithms, on the other hand, have brought excellent results in terms of execution time. In the case of the use of the artificial dataset, the graph approach completed all daily runs in a few minutes (2-3). In the case of the use of the real dataset, instead, the approach to graphs has completed all the daily runs in about 250 min, but considering the 15 days in which the algorithm has been tested, just over 16 min of execution per day would be obtained. This analysis brings the consideration that the algorithm can be thought as pseudo real time, considering the data of the day.

Chapter 7

Conclusions and future developments

In this thesis experiment, different anomaly detection approaches were analyzed and studied within enterprise contests. The anomaly detection approaches used, have led to the construction of two different kinds of algorithms: the one for the detection of hacked company employees and the one for the detection of malicious hosts to which employees have requested connections.

The detection of hacked company users was done mainly through statistical approaches that firstly tried to establish which technique brought the best results, and then to self-improve the classification through appropriate reinforcement mechanisms. The statistical techniques experimented were the Percentile-99, the Z-Score and the Inter-Quartile Range, all used for the detection of anomalies in the connections of users within the company network. The best results were brought by IQR.

The reinforcement approaches, on the other hand, were three. The first with a double reinforcement approach whereby, based on the previous day's results, an attempt was made to widen or narrow a certain coefficient of anomaly discrimination or to increase or decrease the number of anomalies to be taken into account to classify a user as hacked. A second approach, an enhancement of the first, that took into account only certain types of anomalies rather than others. Finally, the

last approach, which through a dynamic system of weights associated with each feature in the dataset, was able to give the best results and denoted some prominence in the features in finding anomalies. The best results, as reported previously, were from the third approach.

Other approaches studied in this thesis work, for the detection of malicious hosts connected with users belonging to the company, made use of Big Data processes and graph algorithms.

Through the Spark framework, and GraphX, it was possible to build a graph that simulated the connections of company users to external hosts. Constructed daily, with therefore all the relative changes, this graph has been considered as a dynamic graph, and therefore studied day by day.

In particular, having available a set of malicious hosts, known a priori, it has been experimented an approach to score that would make possible the detection of new hosts, even these malicious. Through the Pregel paradigm, offered by GraphX thanks to its API, daily runs were simulated and the results obtained were reported.

The reported results of the two approaches used, showed how, in security contexts in which there is the detection of any malicious entity, they can be used appropriately and reliably.

All of these techniques used, moreover, have the ability to be easily modified, and adaptable to the use of other datasets that are similar but different than those used.

Possible future developments, since all of these approaches are very dynamic, may involve improvements in the techniques used for user classification, as well as upgrades in feature anomaly detection. Considering the statistical approaches that make use of reinforcement, it is easy to think that the scope for improvement will grow larger as time goes on, as more and more sophisticated reinforcement

techniques can be used.

The graph approach instead, very innovative, could surely find improvements in the calculation of the score or in the classification of hosts, but the idea behind the algorithm itself, could also be used in different fields like banking, business or health.

List of Figures

List of Tables

Bibliography

[1] *FBI IC3 Report*. 2020. URL: https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf.

[2] *Rapporto Clusit*. 2021. URL: https://clusit.it/rapporto-clusit/.

[3] Mahbubul Alam. *Z-score for anomaly detection*. Ed. by Medium. Sept. 2020. URL: https://towardsdatascience.com/z-score-for-anomaly-detection-d98b0006f510.

[4] *Winsorizing*. URL: https://en.wikipedia.org/wiki/Winsorizing.

[5] *Interquartile range*. URL: https://en.wikipedia.org/wiki/Interquartile_range#Outliers.

[6] Mahbubul Alam. *K-Nearest Neighbors (kNN) for anomaly detection*. Ed. by Medium. Oct. 2020. URL: https://towardsdatascience.com/k-nearest-neighbors-knn-for-anomaly-detection-fdf8ee160d13.

[7] Mahbubul Alam. *Support Vector Machine (SVM) for Anomaly Detection*. Ed. by Medium. Oct. 2020. URL: https://towardsdatascience.com/k-nearest-neighbors-knn-for-anomaly-detection-fdf8ee160d13.

[8] *OneClassSVM*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

[9] Jimeng Sun et al. «Neighborhood formation and anomaly detection in bipartite graphs». In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. 2005, p. 8.

[10] Hanghang Tong and Ching Yung Lin. «Non-negative residual matrix factorization with application to graph anomaly detection». In: *Proceedings of the 11th SIAM International Conference on Data Mining, SDM 2011*. 2011, pp. 143–153.

[11] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. «OddBall: Spotting Anomalies in Weighted Graphs». In: *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*. 2010, pp. 410–421.

[12] Keith Henderson et al. «It's who you know: Graph mining using recursive structural features». In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'11*. 2011, pp. 663–671.

[13] Sergey Brin and Lawrence Page. «The anatomy of a large-scale hypertextual Web search engine». In: *Computer Networks and ISDN Systems* 30.1 (1998), pp. 107–117.

[14] T.H. Haveliwala. «Topic-sensitive PageRank: a context-sensitive ranking algorithm for Web search». In: *IEEE Transactions on Knowledge and Data Engineering* 15.4 (2003), pp. 784–796.

[15] Jing Gao et al. «On Community Outliers and their Efficient Detection in Information Networks». In: July 2010, pp. 813–822.

[16] Yvonne Müller Emmanuel Müller Patricia Iglesias and Böhm Klemens. «Ranking outlier nodes in subspaces of attributed graphs». In: (2013).

[17] Caleb C. Noble and Diane J. Cook. «Graph-Based Anomaly Detection». In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2003, pp. 631–636.

[18] H Bunke et al. *A graph-theoretic approach to enterprise network dynamics*. 2007.

[19] Peter Shoubridge et al. «Detection of Abnormal Change in a Time Series of Graphs». In: *Journal of Interconnection Networks* (2002), pp. 85–101.

[20] Tsuyoshi Idé and Hisashi Kashima. «Eigenspace-Based Anomaly Detection in Computer Systems». In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2004, pp. 440–449.

[21] Keisuke Ishibashi et al. «Detecting anomalous traffic using communication graphs». In: *Telecommunications: The Infrastructure for the 21st Century (WTC)* (2010), pp. 1–6.

[22] J. Sun et al. «GraphScope: Parameter-free mining of large time-evolving graphs». In: 2007, pp. 687–696.

[23] Leto Peel and Aaron Clauset. «Detecting change points in the large-scale structure of evolving networks». In: *CoRR* (2014).

[24] Carey E. Priebe et al. «Scan Statistics on Enron Graphs». In: *Computational and Mathematical Organization Theory* (2005), pp. 229–247.

[25] Misael Mongiovi et al. «Netspot: Spotting significant anomalous regions on dynamic networks». In: 2013.

[26] Shivam Chaudhary. *Why "1.5" in IQR Method of Outlier Detection?* Ed. by Medium. Sept. 2019. URL: `https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097`.

[27] *Python*. URL: `https://www.python.org/`.

[28] *Pandas Library*. URL: `https://pandas.pydata.org/`.

[29] *NumPy Library*. URL: `https://numpy.org/`.

[30] *Matplotlib Library*. URL: `https://matplotlib.org/`.

[31] *Apache Spark*. URL: `https://spark.apache.org/`.

[32] *GraphX*. URL: `https://spark.apache.org/graphx/`.

[33] Grzegorz Malewicz et al. «Pregel: A System for Large-Scale Graph Processing». In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 2010, pp. 135–146. URL: `https://doi.org/10.1145/1807167.1807184`.